On the Convergence of Cloud Computing and Desktop Grids

Presented by Derrick Kondo

Many Slides by

Jeff Barr, Amazon Inc.

and Jeff Dean, Sanjay Ghemawat, Google, Inc.

Outline

- Cloud Computing
  - Background
  - Architecture
  - Map-Reduce
- Desktop Grids
  - Background & contract with clouds
  - Architecture
  - Prediction

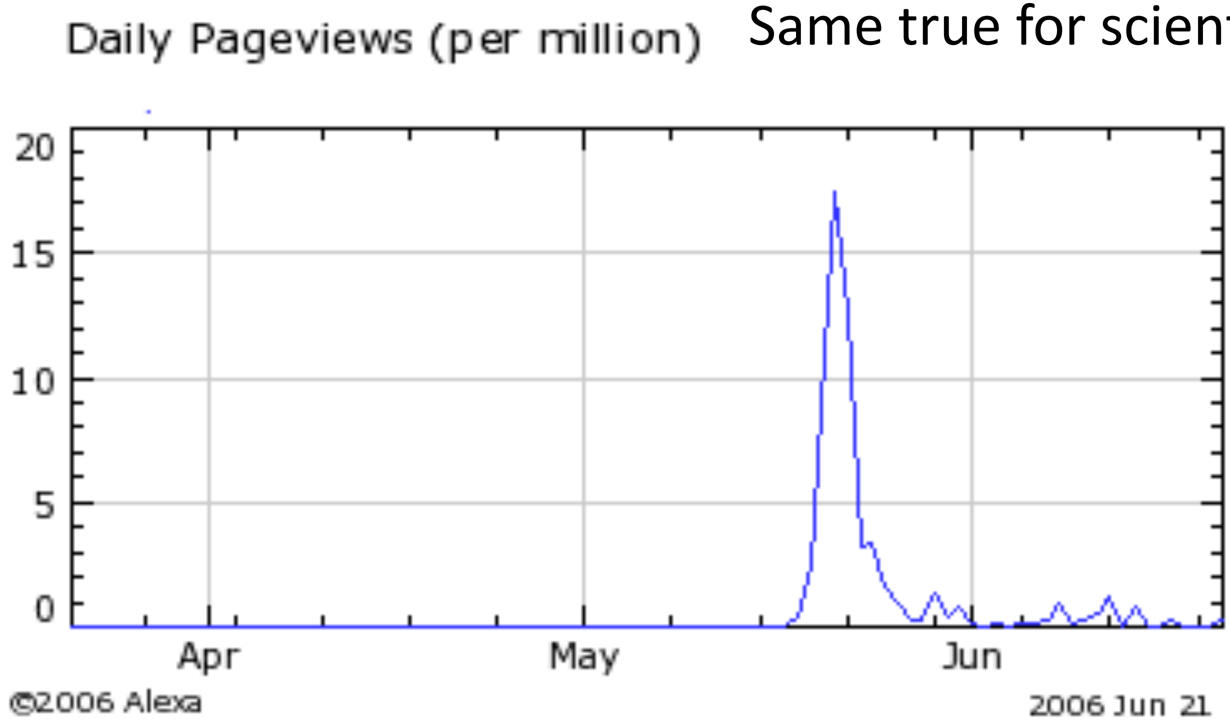
Motivation

- 70% of Web Development Effort is “Muck”:
  - Data Centers
  - Bandwidth / Power / Cooling
  - Operations
  - Staffing
- Scaling is Difficult and Expensive:
  - Large Up-Front Investment
  - Invest Ahead of Demand
  - Load is Unpredictable

Dream or Nightmare?

- Slashdot/Digg/TechCrunch Effect
- Rapid, unexpected customer demand/growth

Same true for scientific workloads



Solution: Cloud Computing

- Scale capacity on demand
- Turn fixed costs into variable costs
- Always available
- Rock-solid reliability
- Simple APIs and conceptual models
- Cost-effective
- Reduced time to market
- Focus on product & core competencies

What is a cloud?

- Cloud computing is Internet-based ("cloud") development and use of computer technology ("computing"). -- Wikipedia
- *A* cloud is a distributed system where the user doesn't care exactly what resources are used to carry out an operation -- Prof. Douglas Thain
- "A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider åand consumers." -- Prof Raj Buyya

Cloud Providers

- Large-scale centralized systems
  - Low reliability, low-cost commodity components
- Google
  - 100,000 systems in 15 data centers [2005]
    - Recent estimate: 500,000 systems in 30 data centers

Figure 5: Sun Microsystems Black Box

1,152 systems in 20x8x8 foot container

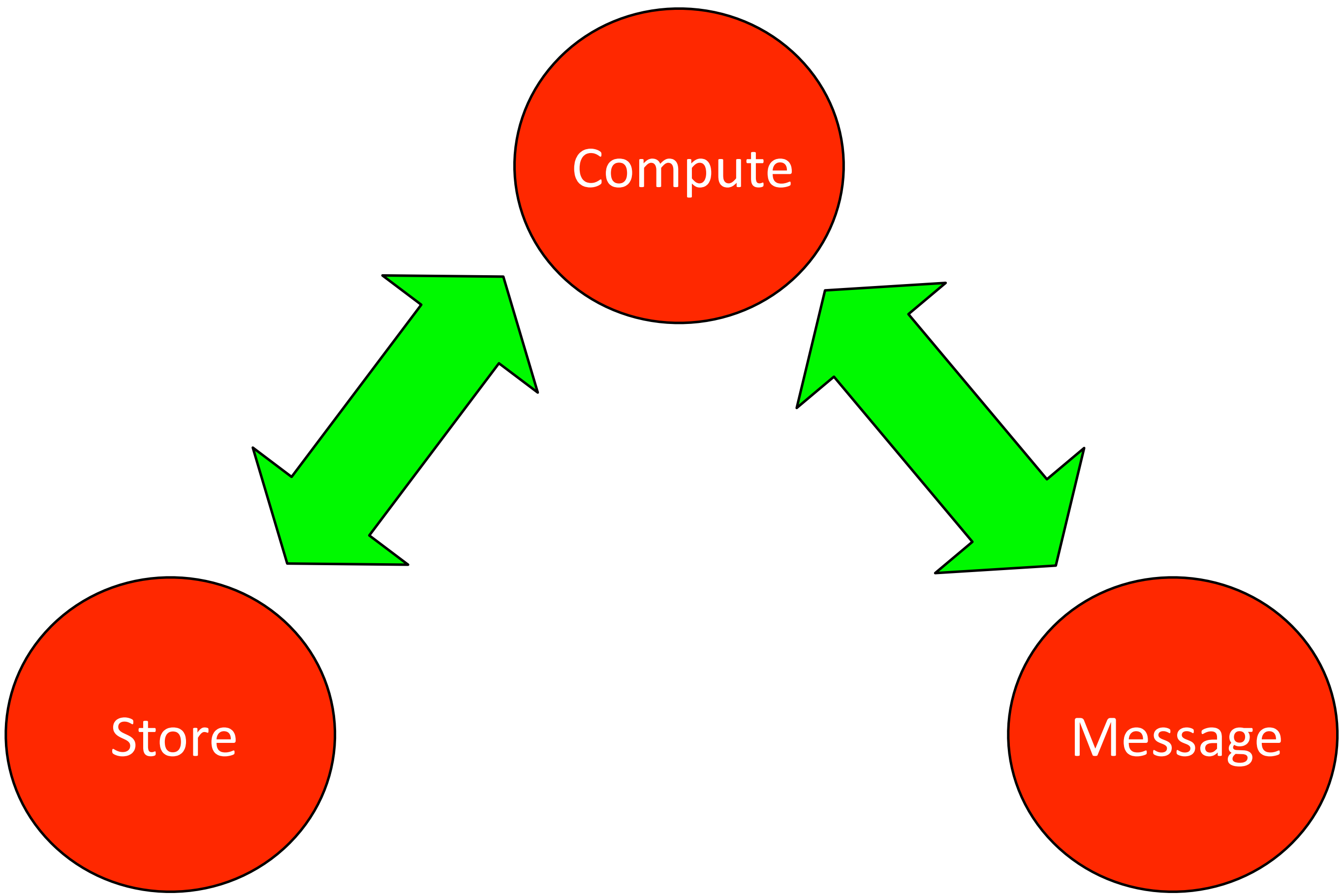
Types of Clouds

- Platform-as-a-service
  - E.g. Amazon's EC2
- Applications-as-a-service
  - E.g. Google App Engine (DataStore/GQL, MapReduce)

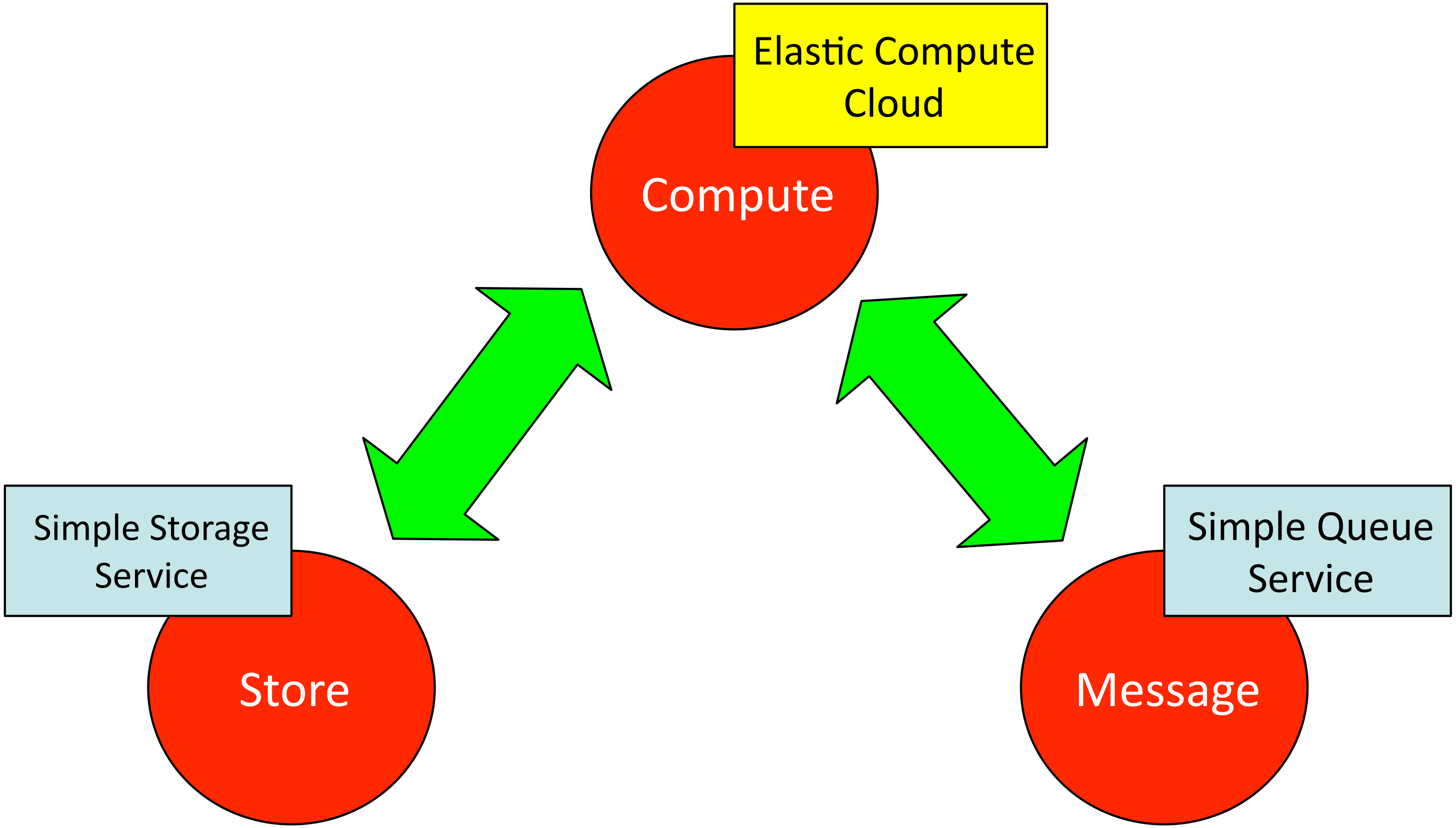
Google App Engine

- Run web applications (Python-based)
- API for data store, google accounts, URL fetching, image manip., email
- Web-based admin console
- Free with up to 500MB of storage and 5 million views

Infrastructure Services



Infrastructure Services



Amazon Simple Storage Service

- 1 B – 5 GB / object
- Fast, Reliable, Scalable
- Redundant, Dispersed
- 99.99% Availability

Goal

- Private or Public
- Per-object URLs & ACLs
- BitTorrent Support

Pricing in Europe

Storage

*** $0.180 per GB – first 50 TB / month of storage used**

* $0.170 per GB – next 50 TB / month of storage used
* $0.160 per GB – next 400 TB / month of storage used
* $0.150 per GB – storage used / month over 500 TB

Data Transfer

*** $0.100 per GB – all data transfer in**

*** $0.170 per GB – first 10 TB / month data transfer out**
* $0.130 per GB – next 40 TB / month data transfer out
* $0.110 per GB – next 100 TB / month data transfer out
* $0.100 per GB – data transfer out / month over 150 TB

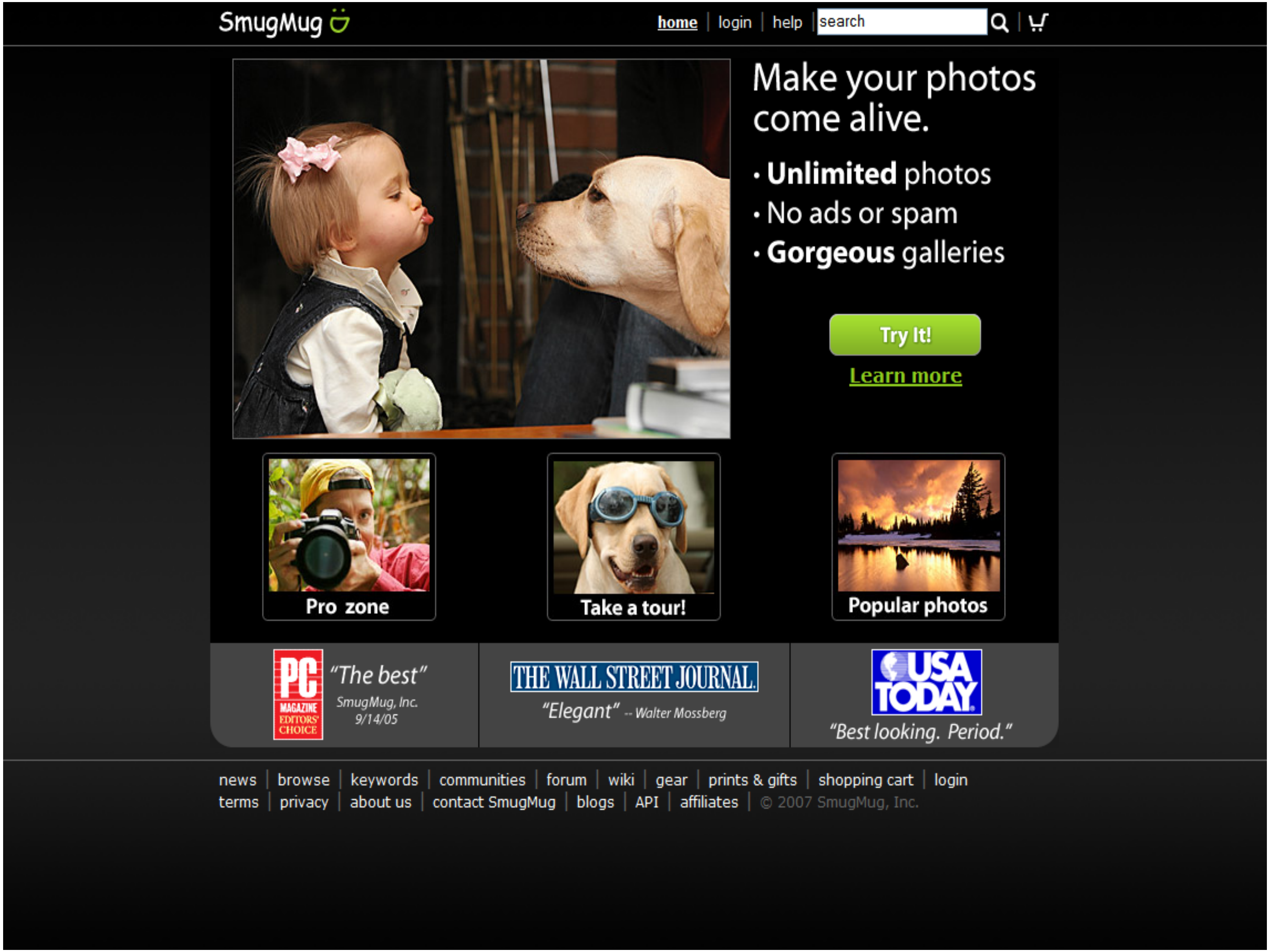
Requests

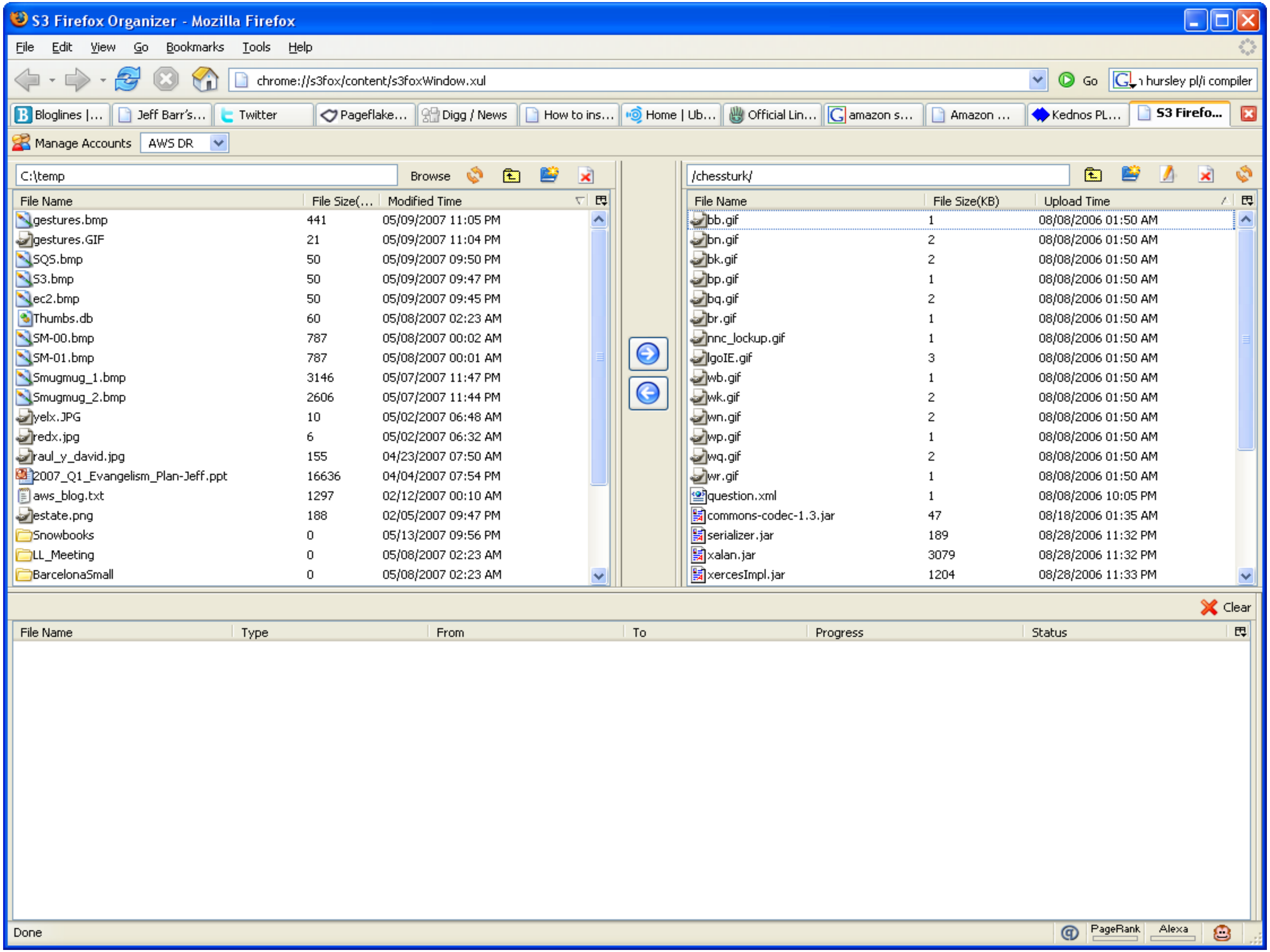
*** $0.012 per 1,000 PUT, COPY, POST, or LIST requests**
*** $0.012 per 10,000 GET and all other requests***

Amazon S3 Concepts

- Objects:
  - Opaque data to be stored (1 byte … 5 Gigabytes)
  - Metadata (attribute-value, up to 4KB)
  - Authentication and access controls
- Buckets (like directories):
  - Object container – any number of objects
  - 100 buckets per account / buckets are “owned”
- Keys:
  - Unique object identifier within bucket
  - Up to 1024 bytes long
  - Flat object storage model
- Functionality
  - – Simple put/get functionality
  - – Limited search functionality
  - – Objects are immutable, cannot be renamed
- Standards-Based Interfaces:
  - REST and SOAP
  - URL-Addressability – every object has a URL

2-level namespace


SmugMug
home | login | help | search
Make your photos come alive.
• Unlimited photos
• No ads or spam
• Gorgeous galleries
Try It!
Learn more
Pro zone
Take a tour!
Popular photos
PC MAGAZINE EDITORS' CHOICE
"The best"
SmugMug, Inc. 9/14/05
THE WALL STREET JOURNAL.
"Elegant" -- Walter Mossberg
USA TODAY
"Best looking. Period."
news | browse | keywords | communities | forum | wiki | gear | prints & gifts | shopping cart | login
terms | privacy | about us | contact SmugMug | blogs | API | affiliates | © 2007 SmugMug, Inc.


S3 Firefox Organizer - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
chrome://s3fox/content/s3foxWindow.xul
Go
Bloglines |...
Jeff Barr's...
Twitter
Pageflake...
Digg / News
How to ins...
Home | Ub...
Official Lin...
amazon s...
Amazon ...
Kednos PL...
S3 Firefo...
Manage Accounts
AWS DR
C:\temp
Browse
/chessturk/
File Name
File Size(...
Modified Time
gestures.bmp 441 05/09/2007 11:05 PM
gestures.GIF 21 05/09/2007 11:04 PM
SQS.bmp 50 05/09/2007 09:50 PM
S3.bmp 50 05/09/2007 09:47 PM
ec2.bmp 50 05/09/2007 09:45 PM
Thumbs.db 60 05/08/2007 02:23 AM
SM-00.bmp 787 05/08/2007 00:02 AM
SM-01.bmp 787 05/08/2007 00:01 AM
Smugmug_1.bmp 3146 05/07/2007 11:47 PM
Smugmug_2.bmp 2606 05/07/2007 11:44 PM
yelx.JPG 10 05/02/2007 06:48 AM
redx.jpg 6 05/02/2007 06:32 AM
raul_y_david.jpg 155 04/23/2007 07:50 AM
2007_Q1_Evangelism_Plan-Jeff.ppt 16636 04/04/2007 07:54 PM
aws_blog.txt 1297 02/12/2007 00:10 AM
estate.png 188 02/05/2007 09:47 PM
Snowbooks 0 05/13/2007 09:56 PM
LL_Meeting 0 05/08/2007 02:23 AM
BarcelonaSmall 0 05/08/2007 02:23 AM
File Name
File Size(KB)
Upload Time
bb.gif 1 08/08/2006 01:50 AM
bn.gif 2 08/08/2006 01:50 AM
bk.gif 2 08/08/2006 01:50 AM
bp.gif 1 08/08/2006 01:50 AM
bq.gif 2 08/08/2006 01:50 AM
br.gif 1 08/08/2006 01:50 AM
nnc_lockup.gif 1 08/08/2006 01:50 AM
lgoIE.gif 3 08/08/2006 01:50 AM
wb.gif 1 08/08/2006 01:50 AM
wk.gif 2 08/08/2006 01:50 AM
wn.gif 2 08/08/2006 01:50 AM
wp.gif 1 08/08/2006 01:50 AM
wq.gif 2 08/08/2006 01:50 AM
wr.gif 1 08/08/2006 01:50 AM
question.xml 1 08/08/2006 10:05 PM
commons-codec-1.3.jar 47 08/18/2006 01:35 AM
serializer.jar 189 08/28/2006 11:32 PM
xalan.jar 3079 08/28/2006 11:32 PM
xercesImpl.jar 1204 08/28/2006 11:33 PM
Clear
File Name
Type
From
To
Progress
Status
Done
PageRank
Alexa

Amazon Elastic Compute Cloud

EC2

Amazon EC2

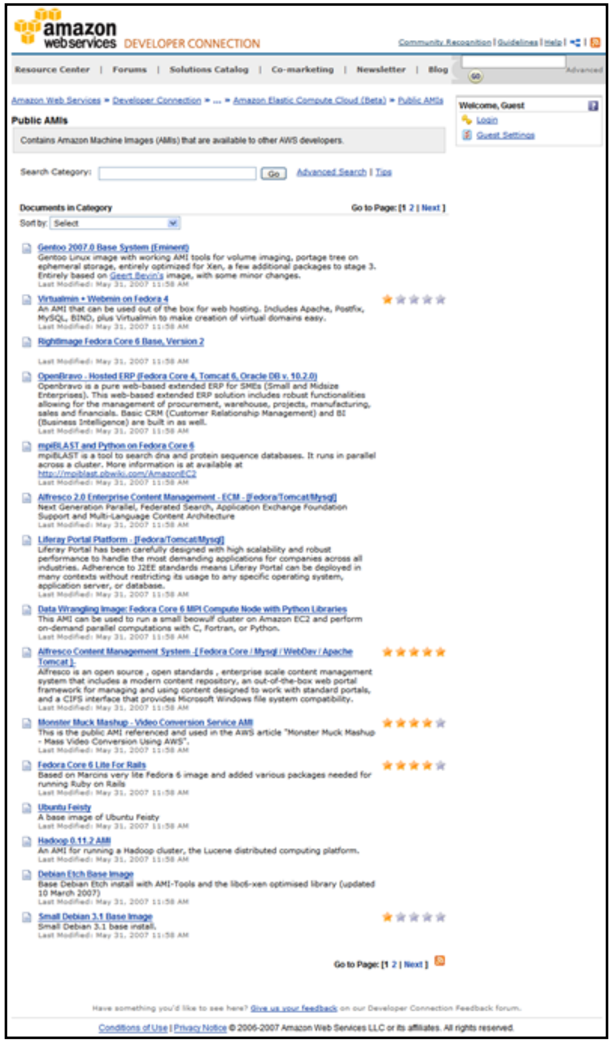
- Virtual environment for linux/windows applications
  - Create Amazon Machine Image (AMI) with app's, lib's, data, config settings,
  - Upload image to S3, then start/stop/monitor images

Amazon EC2 Features

- Elastic: can increase number of resources as needed
- Configurability: can configure hard resources (as instances) or software stack: OS, lib's, app's with root access
- Reliability: 99.99%
- For applications
  - Persistant storage (independant of life of instance)
  - Multiple locations: availability zones
  - Static IP addresses associated with account (not instance)
    - Can remap IP addresses to another instance or availability zone as needed

Amazon EC2 Concepts

- Amazon Machine Image (AMI):
  - Bootable root disk
  - Pre-defined or user-built
  - Catalog of user-built AMIs
- Instance:
  - Running copy of an AMI
  - Launch in less than 2 minutes
  - Start/stop programmatically
- Network Security Model:
  - Explicit access control
  - Security groups
- Inter-service bandwidth is free



Standard Instances

- Small Instance (Default) 1.7 GB of memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB of instance storage, 32-bit platform
- Large Instance 7.5 GB of memory, 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 850 GB of instance storage, 64-bit platform
- Extra Large Instance 15 GB of memory, 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform
- EC2 Compute Unit (ECU) – One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

Large instances

- Instances of this family have proportionally more CPU resources than memory (RAM) and are well suited for compute-intensive applications.
- High-CPU Medium Instance 1.7 GB of memory, 5 EC2 Compute Units (2 virtual cores with 2.5 EC2 Compute Units each), 350 GB of instance storage, 32-bit platform
- High-CPU Extra Large Instance 7 GB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform

Operating Systems and Software

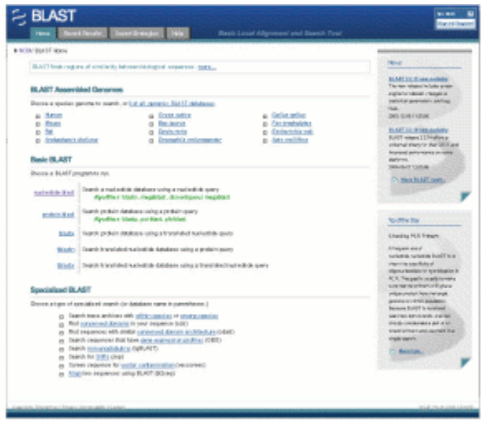
- Operating Systems
  - Red Hat Enterprise Linux Windows Server 2003 Oracle Enterprise Linux
  - OpenSolaris openSUSE Linux Ubuntu Linux
  - Fedora Gentoo Linux Debian
- Software
  - Databases
    - Oracle 11g, MySQL Enterprise, Microsoft SQL Server Standard 2005
  - Batch Processing
    - Hadoop, Condor
  - eb Hosting
    - Apache HTTP, IIS/Asp.Net

Pricing

- Pay as you use
- Standard Instances
  - Linux
    - Small (Default) $0.10 per hour
    - Large $0.40 per hour
    - Extra Large $0.80 per hour
- High CPU Instances
  - Medium $0.20 per hour
  - Extra Large $0.80 per hour
- Internet Data Transfer
  - Data transfer in: $0.10 per GB
  - Data transfer out: $0.17 per GB

Amazon EC2 At Work

- Startups
  - Cruxy – Media transcoding
  - GigaVox Media – Podcast Management
- Fortune 500 clients:
  - High-Impact, Short-Term Projects
  - Development Host
- Science / Research:
  - Hadoop / MapReduce
  - mpiBLAST
- Load-Management and Load Balancing Tools:
  - Pound
  - Weogeo
  - Rightscale



Can Clouds Work for Science?

- Applications don't need durability, availability, and access performance all bundled together

CPU costs dominate for scientific workflow application called montage

Table 2. The resources needed to provide high performance data access, high data availability and long data durability are different

Characteristics	**Resources and techniques to provide them**
High-performance data access	Geographical data (or storage) replication to improve access locality, high-speed storage, fat networks
Durability	Data replication - possible at various levels: hardware (RAID), multiple locations, multiple media; erasure codes
Availability	Server/service replication, hot-swap technologies, multi hosting, techniques to increase availability for auxiliary services (e.g., authentication, access control)

Table 3. Application classes and their associated requirements

Application class	**Durability**	**Availability**	**High access speed**
Cache	No	Depends	Yes
Long-term archival	Yes	No	No
Online production	No	Yes	Yes
Batch production	No	No	Yes

MapReduce:
Simplified Data Processing on

These are slides from Dan Weld's class at U. Washington

(who in turn made his slides based on those by Jeff Dean, Sanjay Ghemawat, Google, Inc.)

- An abstraction is a simple interface that allows you to scale up well-structured problems to run on hundreds or thousands of computers at once.
  - -- Douglas Thain

Large-scale Management Issues

- How to parallelize
- Data distribution
- Scheduling
- Load balancing
- Failure management
- Deployment

MapReduce

- MapReduce provides
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates

Map/Reduce

- Map/Reduce
  - Programming model from Lisp
  - (and other functional languages)
    - state what you want to do not how to get it
- Many problems can be phrased this way
- Easy to distribute across nodes
- Nice retry/failure semantics

Map in Lisp (Scheme)

Map in Lisp (Scheme)

- (map **f list [list_2 list_3 …]**)

Map in Lisp (Scheme)

- (map f list [list$_2$ list$_3$ ...])

Unary operator

Map in Lisp (Scheme)

- (map f **list** [$\textbf{list}_2$ $\textbf{list}_3$ ...])

Unary operator

Map in Lisp (Scheme)

- (map f list [list$_2$ list$_3$ ...])

Unary operator

- (map square '(1 2 3 4))

Map in Lisp (Scheme)

- (map f $list_1$ [$list_2$ $list_3$ …])

Unary operator

- (map square '(1 2 3 4))

Binary operator

Map in Lisp (Scheme)

- (map f list [$list_2$ $list_3$ ...])

Unary operator

- (map square '(1 2 3 4))

Binary operator

Map in Lisp (Scheme)

- (map f list [list$_2$ list$_3$ ...])
- (map square '(1 2 3 4))

Unary operator

Binary operator

Map in Lisp (Scheme)

- (map f list [$list_2$ $list_3$ ...])
- (map square '(1 2 3 4))

Unary operator

Binary operator

Map in Lisp (Scheme)

- (map f list [list$_2$ list$_3$ ...])
- (map square '(1 2 3 4))
  - (1 4 9 16)

Unary operator

Binary operator

Map in Lisp (Scheme)

- (map f list [$list_2$ $list_3$ ...])

Unary operator

- (map square '(1 2 3 4))
  - (1 4 9 16)

Binary operator

Map in Lisp (Scheme)

- (map f list [$list_2$ $list_3$ ...])
- (map square '(1 2 3 4))
  - (1 4 9 16)
- (reduce + '(1 4 9 16))

Unary operator

Binary operator

Map in Lisp (Scheme)

- (map f list [$list_2$ $list_3$ ...])

Unary operator

- (map square '(1 2 3 4))
  - (1 4 9 16)

Binary operator

- (reduce + '(1 4 9 16))
  - (+ 16 (+ 9 (+ 4 1)))

Map in Lisp (Scheme)

- (map f list [$list_2$ $list_3$ ...])

Unary operator

- (map square '(1 2 3 4))
  - (1 4 9 16)

Binary operator

- (reduce + '(1 4 9 16))
  - (+ 16 (+ 9 (+ 4 1)))
  - 30

Map/Reduce ala Google

Map/Reduce ala Google

- map(key, val) is run on each item in set
  - emits new key, val pairs

- reduce(key, vals) is run for each unique key emitted by map()
  - emits final output

count words in docs

count words in docs

- Input consists of (url, contents) pairs

count words in docs

- Input consists of (url, contents) pairs

- map(key=url, val=contents):
  - For each word w in contents, emit (w, “1”)

count words in docs

- Input consists of (url, contents) pairs

- map(key=url, val=contents):
  - For each word w in contents, emit (w, “1”)

- reduce(key=word, values=uniq_counts):

count words in docs

- Input consists of (url, contents) pairs

- map(key=url, val=contents):
  - For each word w in contents, emit (w, “1”)

- reduce(key=word, values=uniq_counts):
  - Sum all “1”s in values list

count words in docs

- Input consists of (url, contents) pairs
- map(key=url, val=contents):
  - For each word w in contents, emit (w, “1”)
- reduce(key=word, values=uniq_counts):
  - Sum all “1”s in values list
  - Emit result “(word, sum)”

Count,

map(key=url, val=contents):

For each word w in contents, emit (w, "1")

reduce(key=word, values=uniq_counts):

Sum all "1"s in values list

see bob throw

see spot run

Count,

map(key=url, val=contents):

    For each word w in contents, emit (w, "1")

reduce(key=word, values=uniq_counts):

    Sum all "1"s in values list

see bob throw
see spot run

see	1
bob	1
run	1
see	1
spot	1
throw	1

Count,

map(key=url, val=contents):

For each word w in contents, emit (w, “1”)

reduce(key=word, values=uniq_counts):

Sum all “1”s in values list

see bob throw

see spot run

see	1
bob	1
run	1
see	1
spot	1
throw	1

bob	1
run	1
see	2
spot	1
throw	1

Grep

Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):

Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, “1”)

Grep

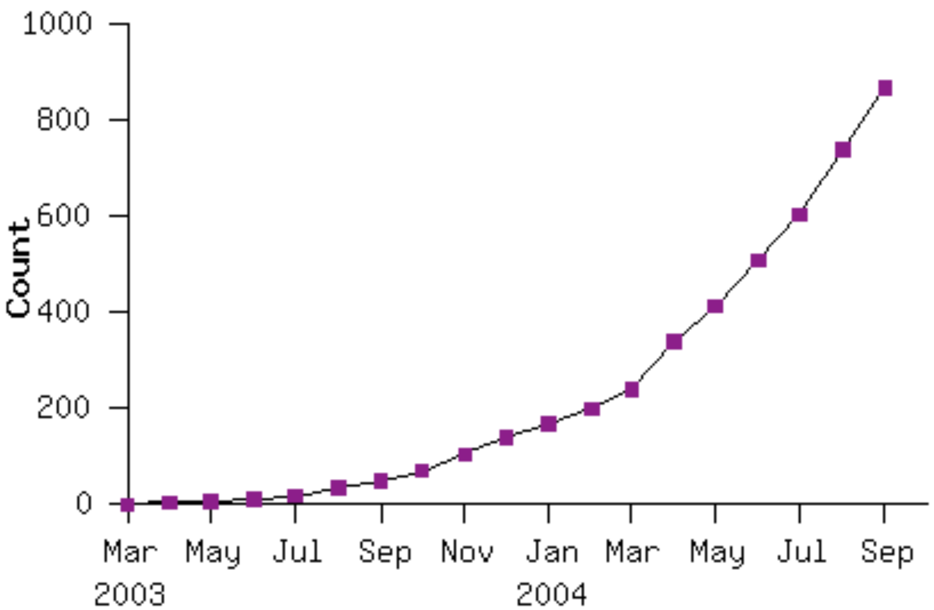
- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, "1")
- reduce(key=line, values=uniq_counts):

Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, “1”)

- reduce(key=line, values=uniq_counts):
  - Don’t do anything; just emit line

Model is Widely Applicable



Example uses:

distributed grep	distributed sort	web link-graph reversal
term-vector / host	web access log stats	inverted index construction
document clustering	machine learning	statistical machine translation

Implementation Overview

Typical cluster:

- 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
- Limited bisection bandwidth
- Storage is on local IDE disks
- GFS: distributed file system manages data (SOSP'03)
- Job scheduling system: jobs made up of tasks, scheduler assigns tasks to machines

Implementation is a C++ library linked into user programs

Execution Overview

Execution Overview

- How is this distributed?

Execution Overview

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel

Execution Overview

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
  2. After all map()s are complete, consolidate all emitted values for each unique emitted key

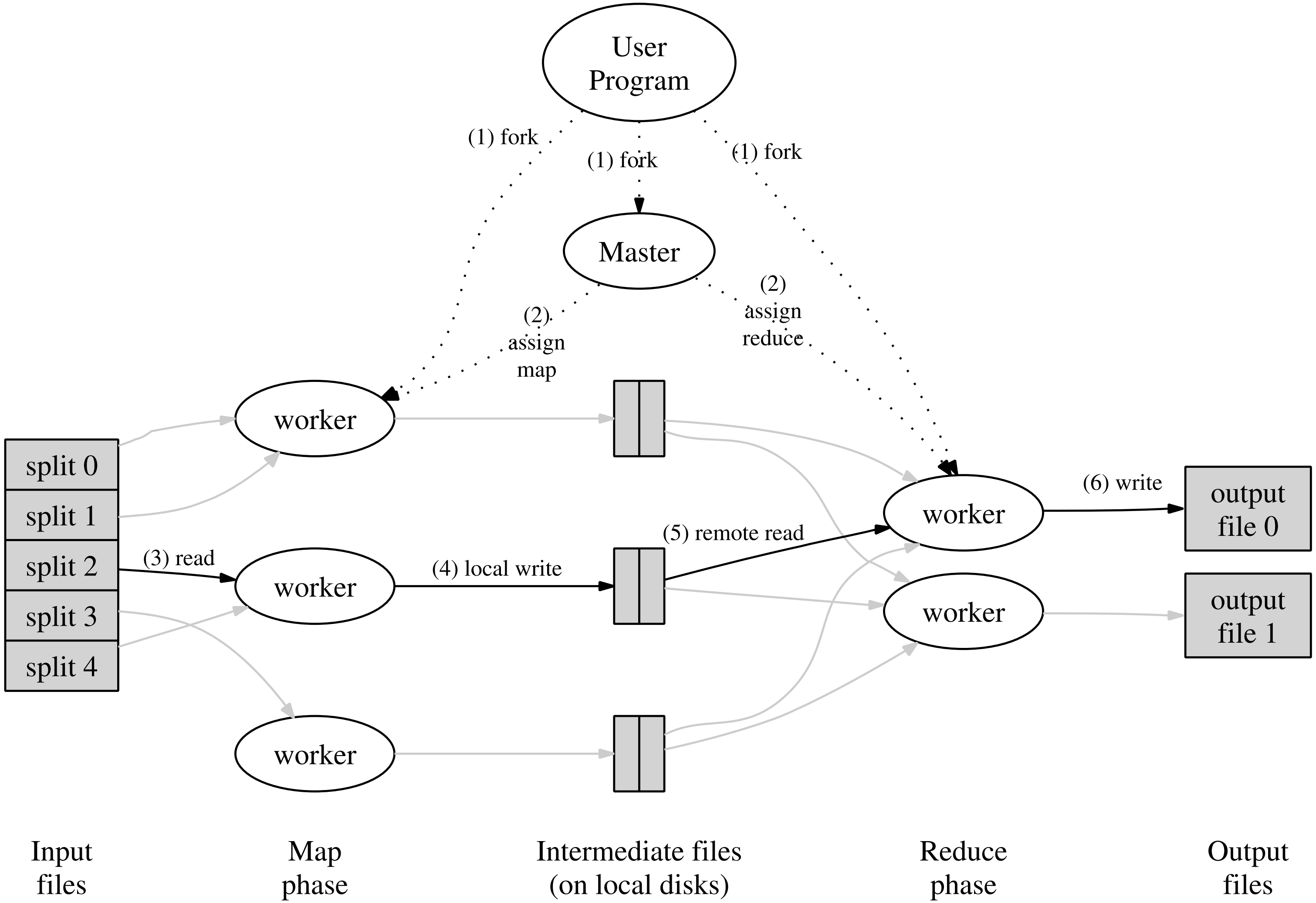
Execution Overview

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
  2. After all map()s are complete, consolidate all emitted values for each unique emitted key
  3. Now partition space of output map keys, and run reduce() in parallel

Execution Overview

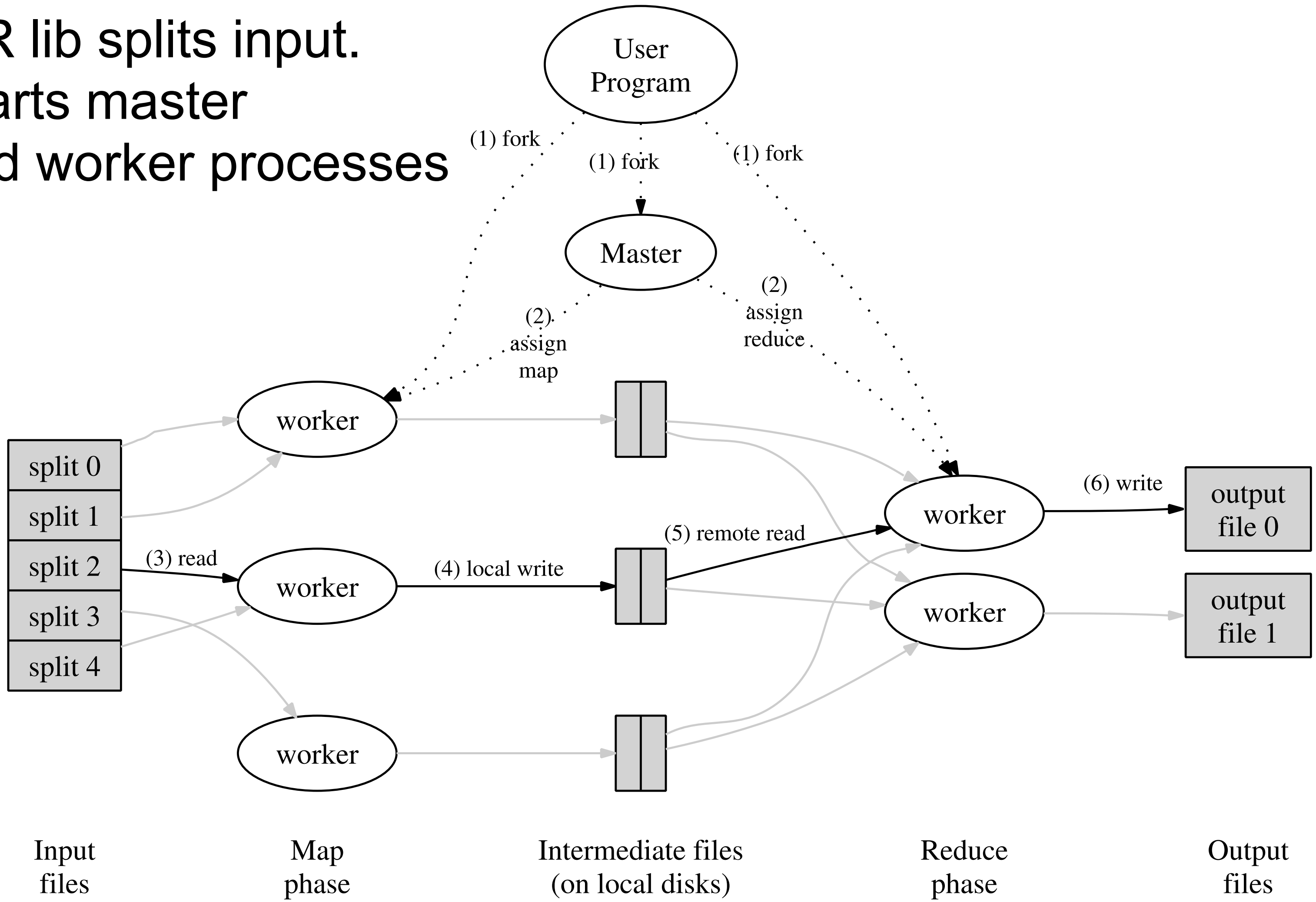
- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
  2. After all map()s are complete, consolidate all emitted values for each unique emitted key
  3. Now partition space of output map keys, and run reduce() in parallel
- If map() or reduce() fails, reexecute!

Execution in more detail



Execution in more detail

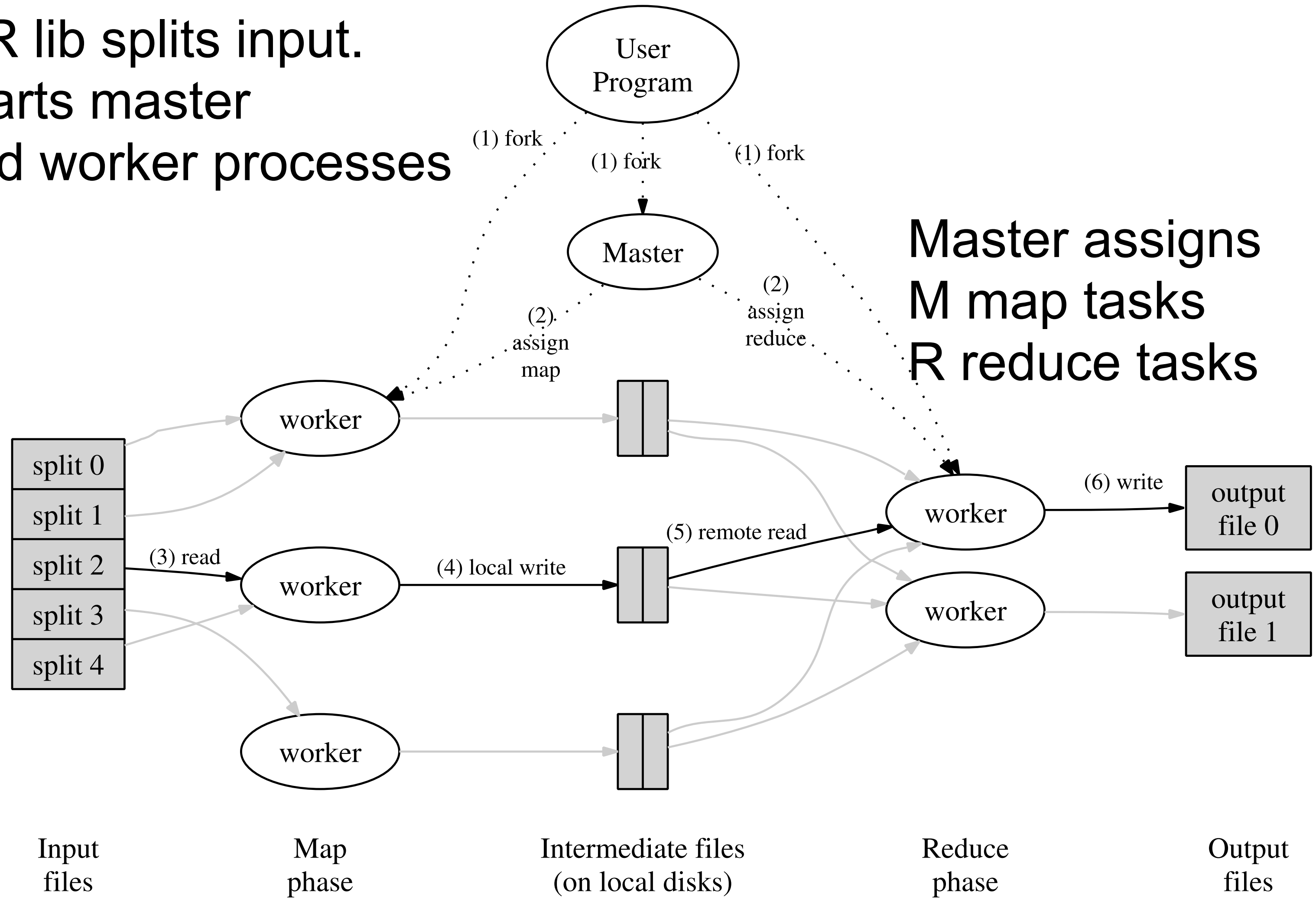
MR lib splits input.
Starts master
and worker processes



Execution in more detail

MR lib splits input.
Starts master
and worker processes

Master assigns
M map tasks
R reduce tasks

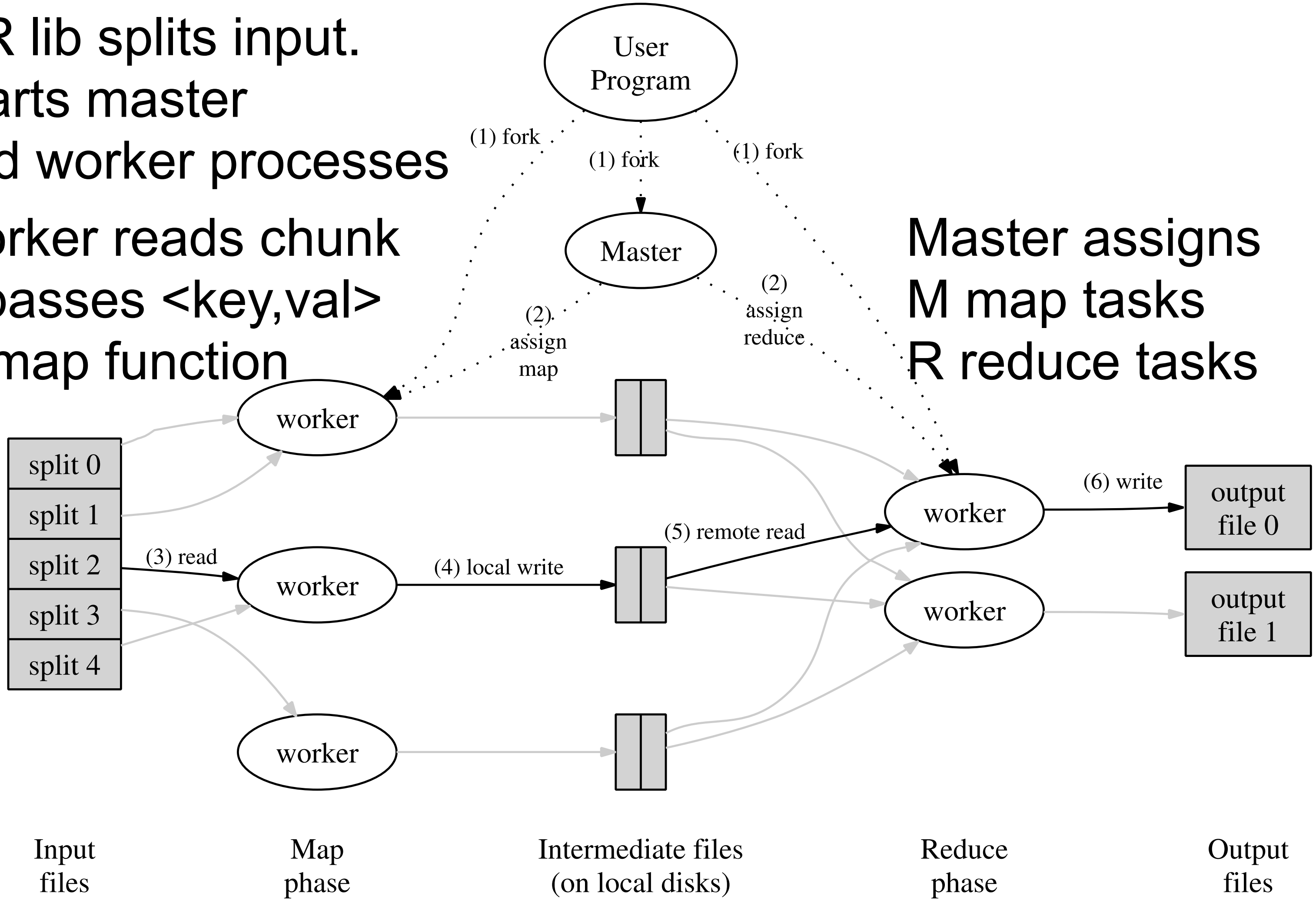


Execution in more detail

MR lib splits input.
Starts master
and worker processes

Worker reads chunk
& passes <key,val>
to map function

Master assigns
M map tasks
R reduce tasks



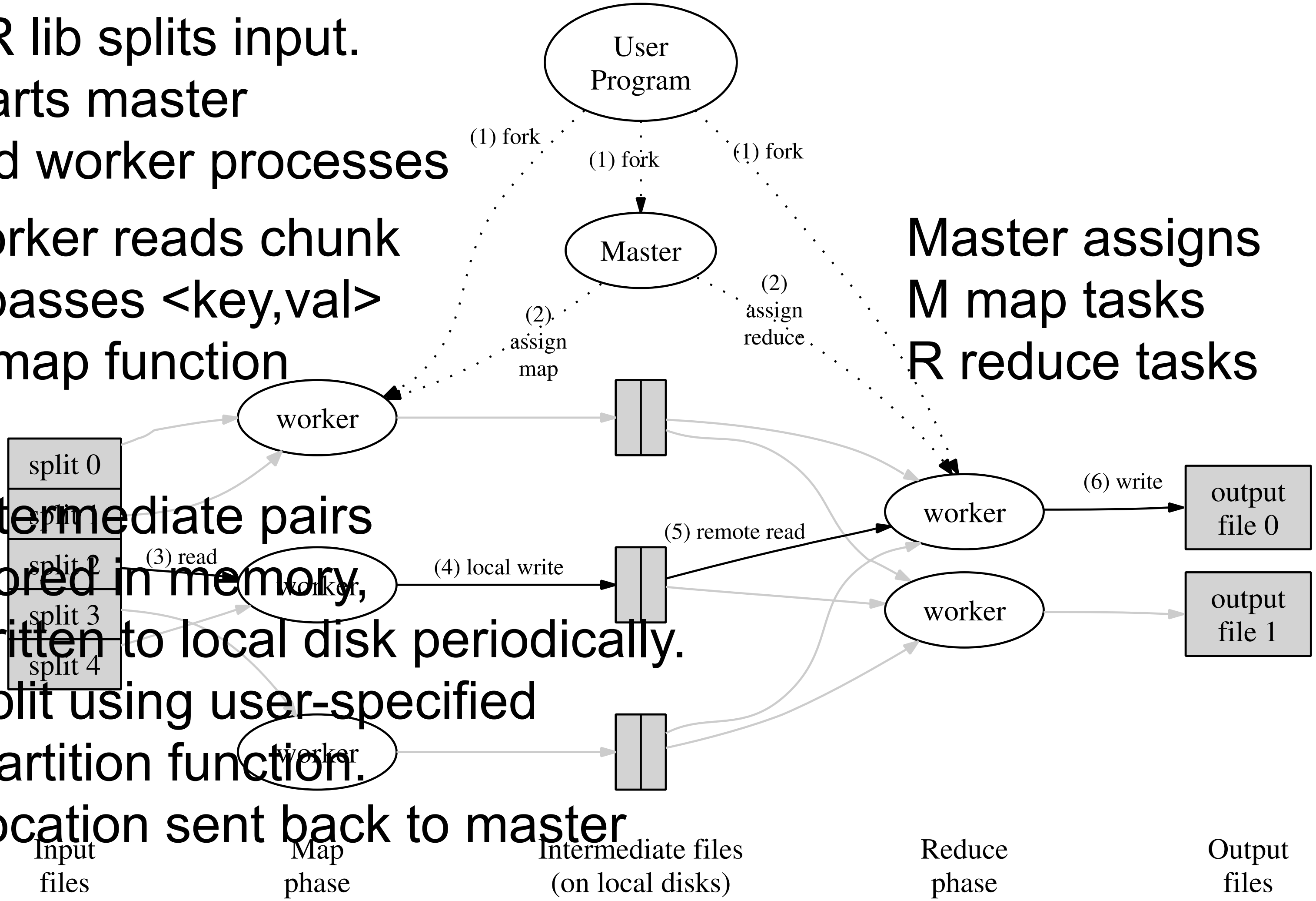
Execution in more detail

MR lib splits input.
Starts master
and worker processes

Worker reads chunk
& passes <key,val>
to map function

Master assigns
M map tasks
R reduce tasks

Intermediate pairs
stored in memory,
written to local disk periodically.
Split using user-specified
 partition function.
Location sent back to master



Execution in more detail

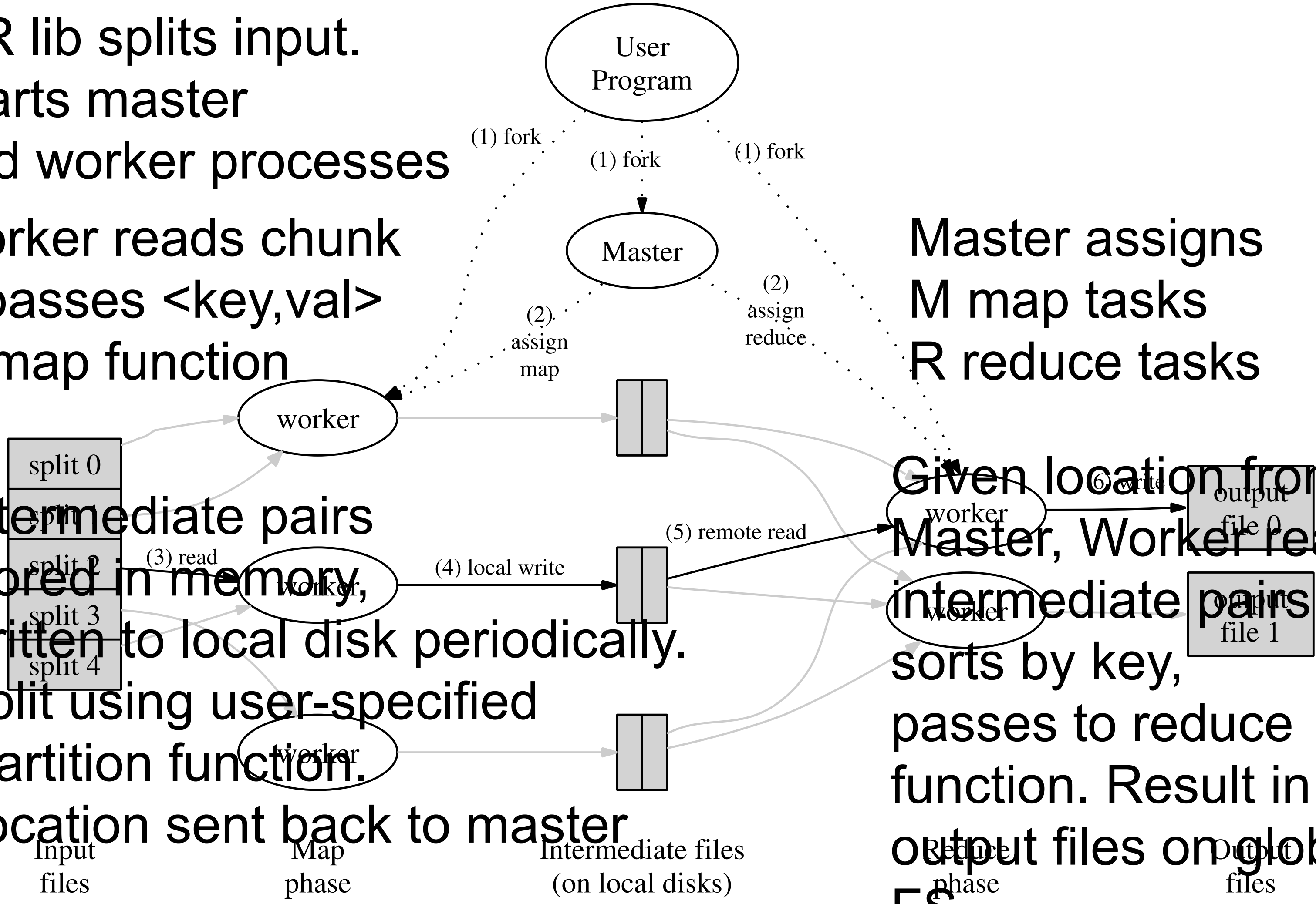
MR lib splits input.
Starts master
and worker processes

Worker reads chunk
& passes <key,val>
to map function

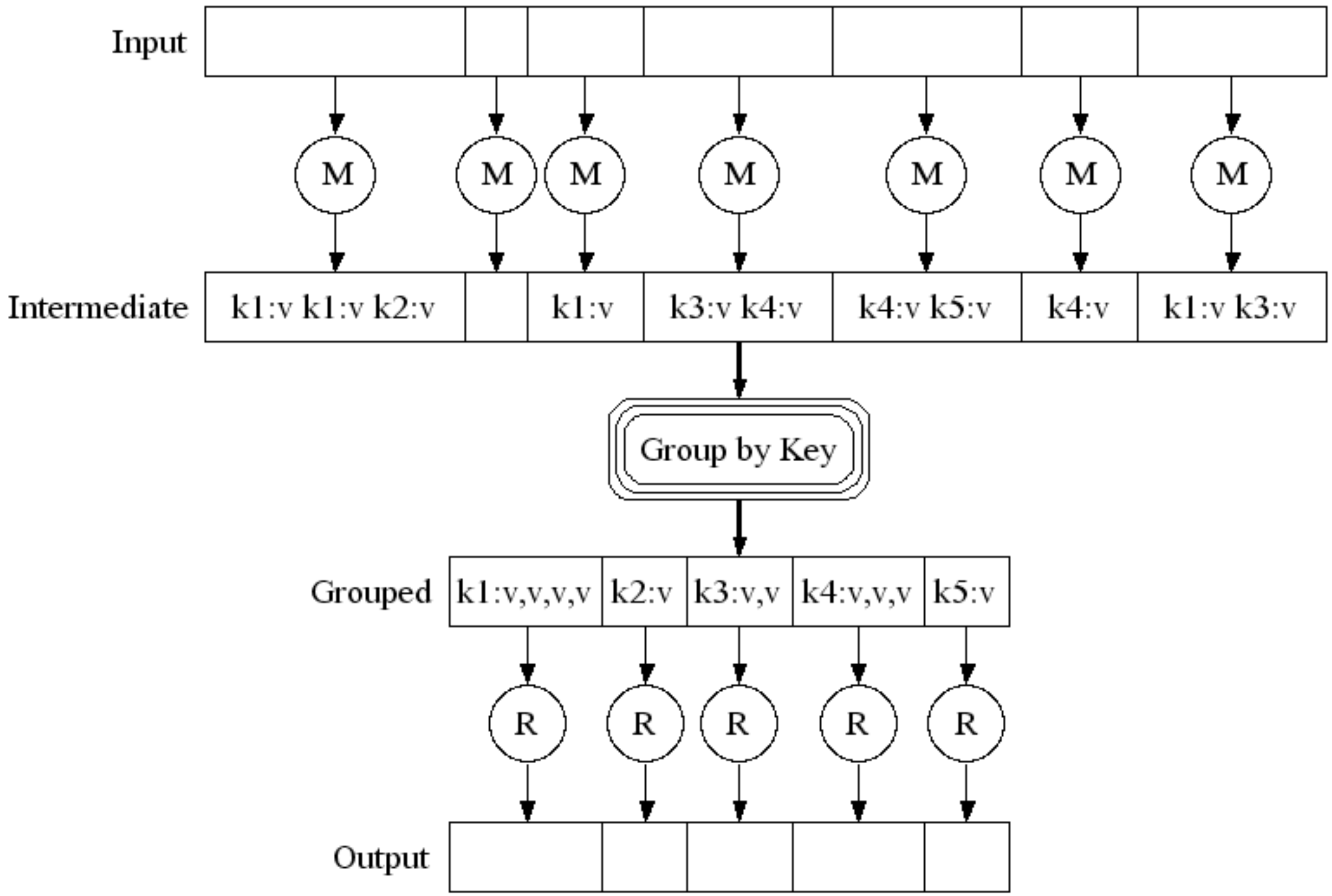
Intermediate pairs
stored in memory,
written to local disk periodically.
Split using user-specified
partition function.
Location sent back to master

Master assigns
M map tasks
R reduce tasks

Given location from
Master, Worker reads
intermediate pairs,
sorts by key,
passes to reduce
function. Result in R
output files on global
FS

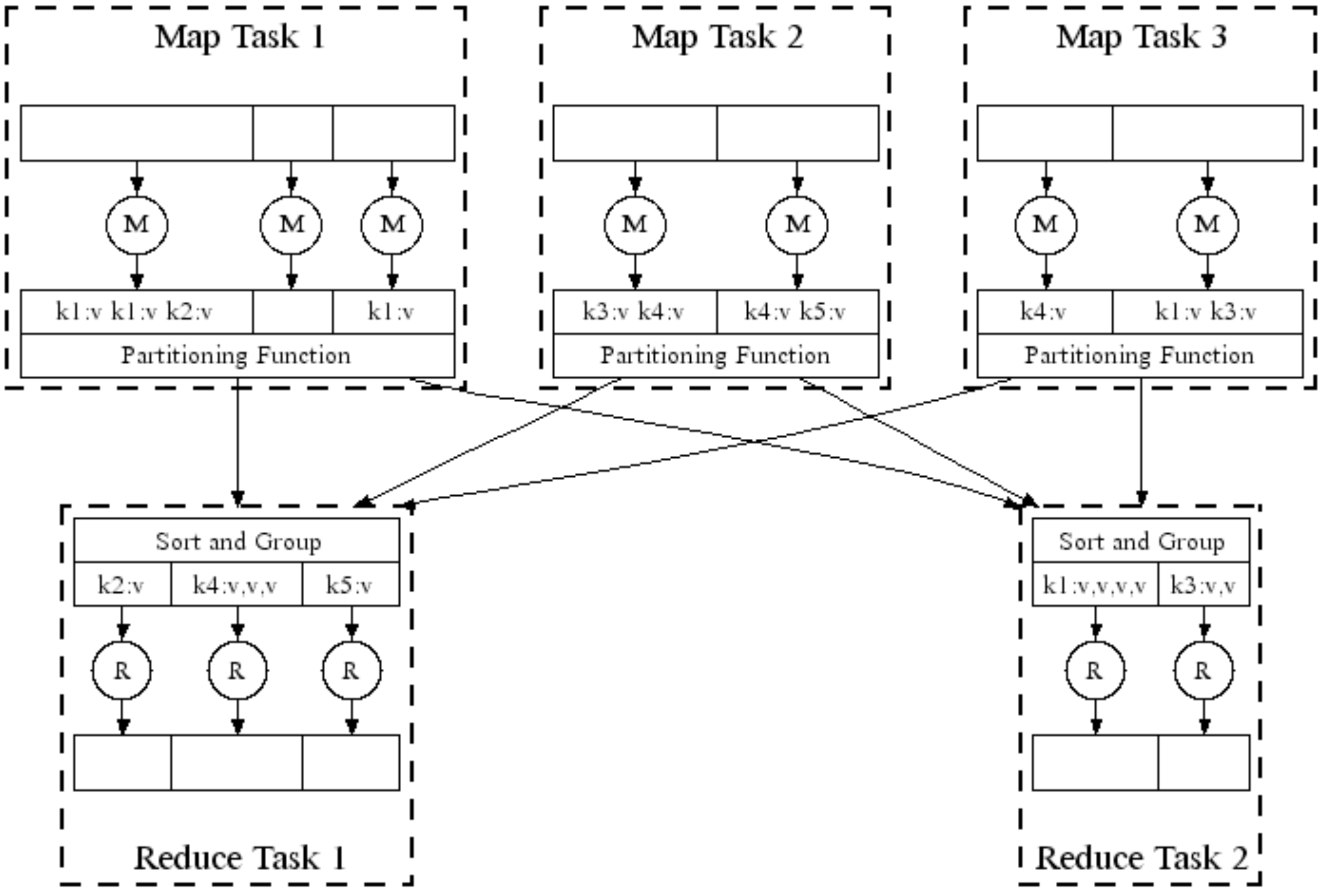


Key Grouping



Parallel Execution

Partition function hashes by key. E.g. hash(key) mod R.



Fault Tolerance / Workers

Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

- Detect failure via periodic heartbeats

Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks

Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why??? (Complete tasks on local disk)

Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why??? (Complete tasks on local disk)
- Re-execute in progress reduce tasks

Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why??? (Complete tasks on local disk)
- Re-execute in progress reduce tasks
- Task completion committed through master

Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why??? (Complete tasks on local disk)
- Re-execute in progress reduce tasks
- Task completion committed through master

Robust: lost 1600/1800 machines once → finished ok

Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why??? (Complete tasks on local disk)
- Re-execute in progress reduce tasks
- Task completion committed through master

Robust: lost 1600/1800 machines once → finished ok

Semantics in presence of failures: see paper

Master Failure

- Could handle, … ?
- But don't yet
  - (master failure unlikely)
  - Could use VM mechanism to hide master failure

Refinement:

Slow workers significantly delay completion time

- Other jobs consuming resources on machine
- Bad disks w/ soft errors transfer data slowly
- Weird things: processor caches disabled (!!)

Solution: Near end of phase, spawn backup tasks

- Whichever one finishes first "wins"

Dramatically shortens job completion time

Refinement
Skipping Bad Records

- Map/Reduce functions sometimes fail for particular inputs
  - Best solution is to debug & fix
    - Not always possible ~ third-party source libraries
  - On segmentation fault:
    - Send UDP packet to master from signal handler
    - Include sequence number of record being processed
  - If master sees two failures for same record:
    - Next worker is told to skip the record

Other Refinements

- Sorting guarantees
  - within each reduce partition
- Compression of intermediate data
- Combiner
  - Useful for saving network bandwidth
- Local execution for debugging/testing
- User-defined counters

Performance

Tests run on cluster of 1800 machines:

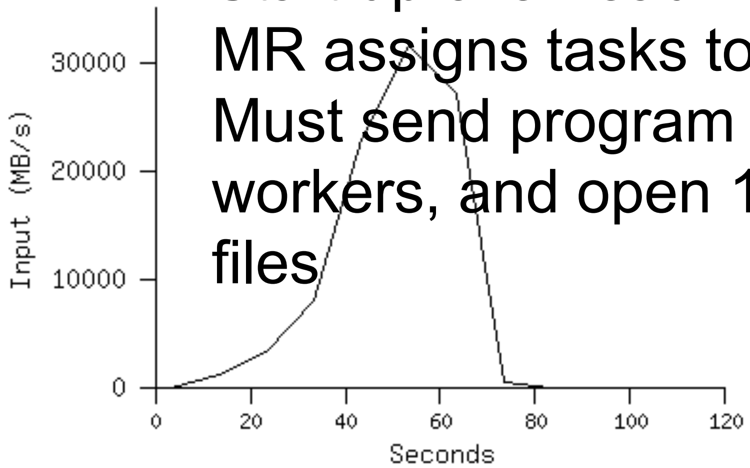
- 4 GB of memory
- Dual-processor 2 GHz Xeons with Hyperthreading
- Dual 160 GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100 Gbps

Two benchmarks:

MR_GrepScan	1010 100-byte records to extract records matching a rare pattern (92K matching records)

MR_Grep

Rate at which input is scanned



Start up overhead high as MR assigns tasks to workers. Must send program to all workers, and open 1000 input files

Time

Locality optimization helps:

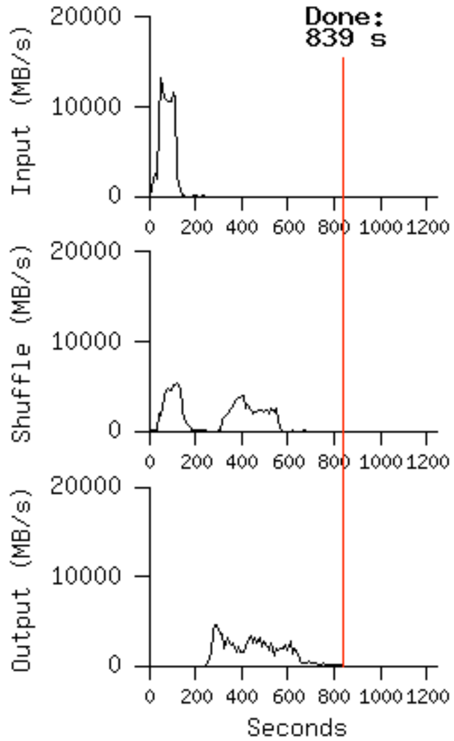
- Input stored on FS in 64GB chunks
  - Workers are spawned near corresponding chucks
- 1800 machines read 1 TB at peak ~31 GB/s
- W/out this, rack switches would limit to 10 GB/s

Startup overhead is significant for short jobs

MR_Sort

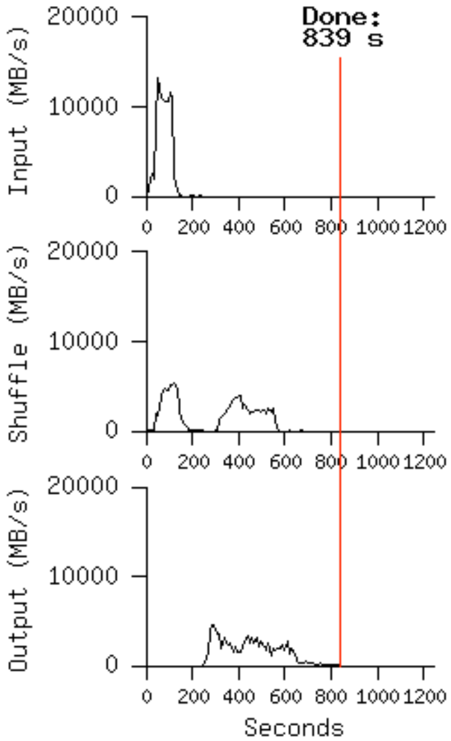
- sort program sorts 1010 100-byte records (approximately 1 terabyte of data)
- map: extract 10-byte sorting key. emit key and line as value
- reduce: built-in identity function
- input data split into 64-MB pieces (M=15000)
- output data in 4000 files (R=4000)
- Partition function uses initial bytes of key to place in one of R chunks
  - Local sort done for each R chunk by MR before the "reduce"
  - Map task send intermediate output to local disk before shuffling to form partition

MR_Sort

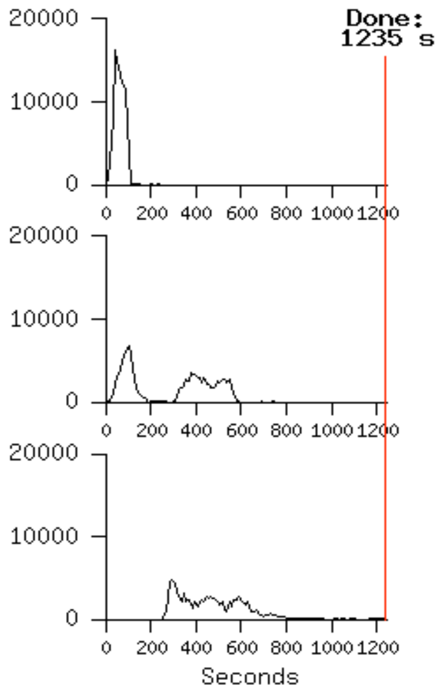

Done:
839 s
Input (MB/s)
Shuffle (MB/s)
Output (MB/s)
20000
10000
0
0 200 400 600 800 1000 1200
Seconds

MR_Sort

Normal

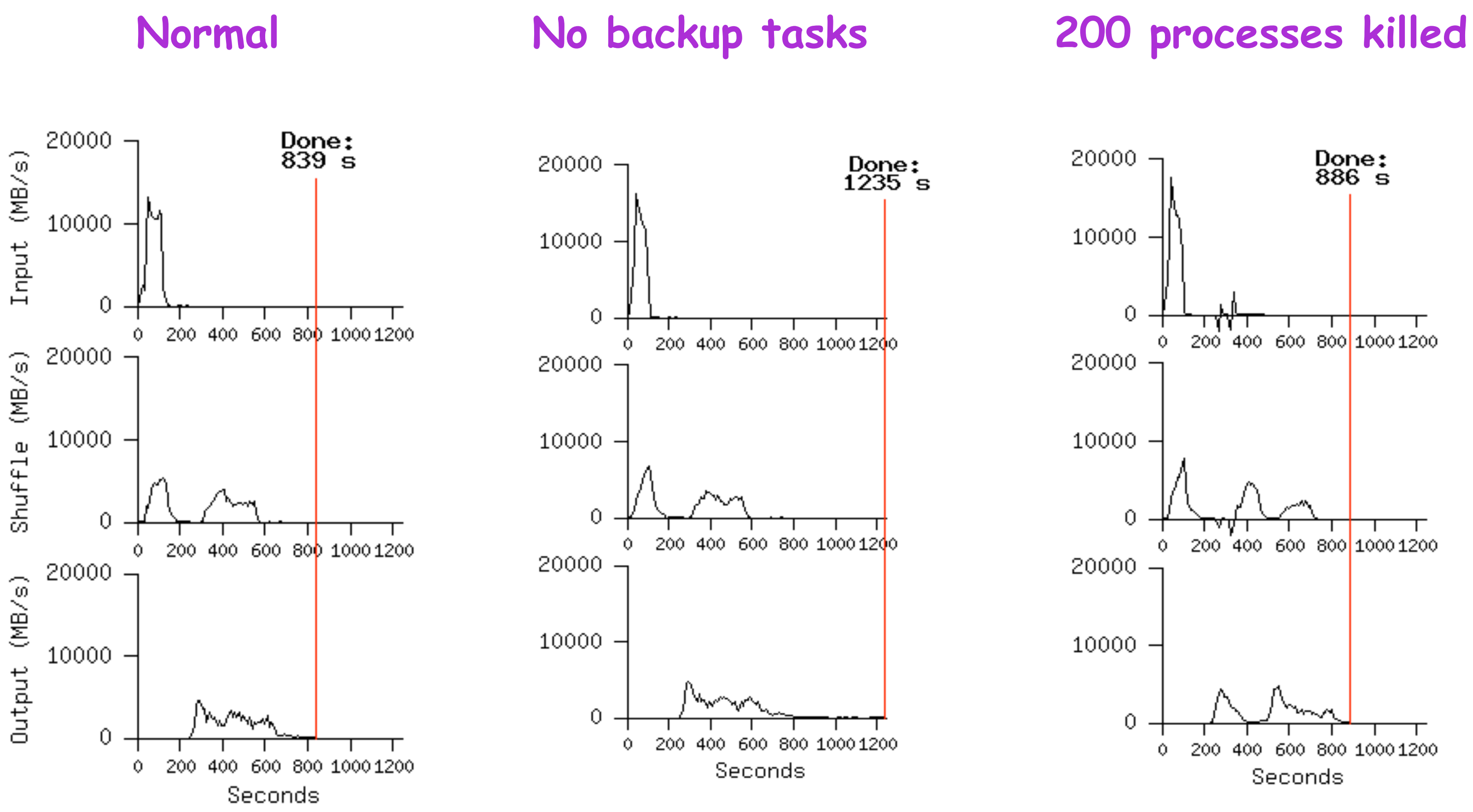


No backup tasks



200 processes killed

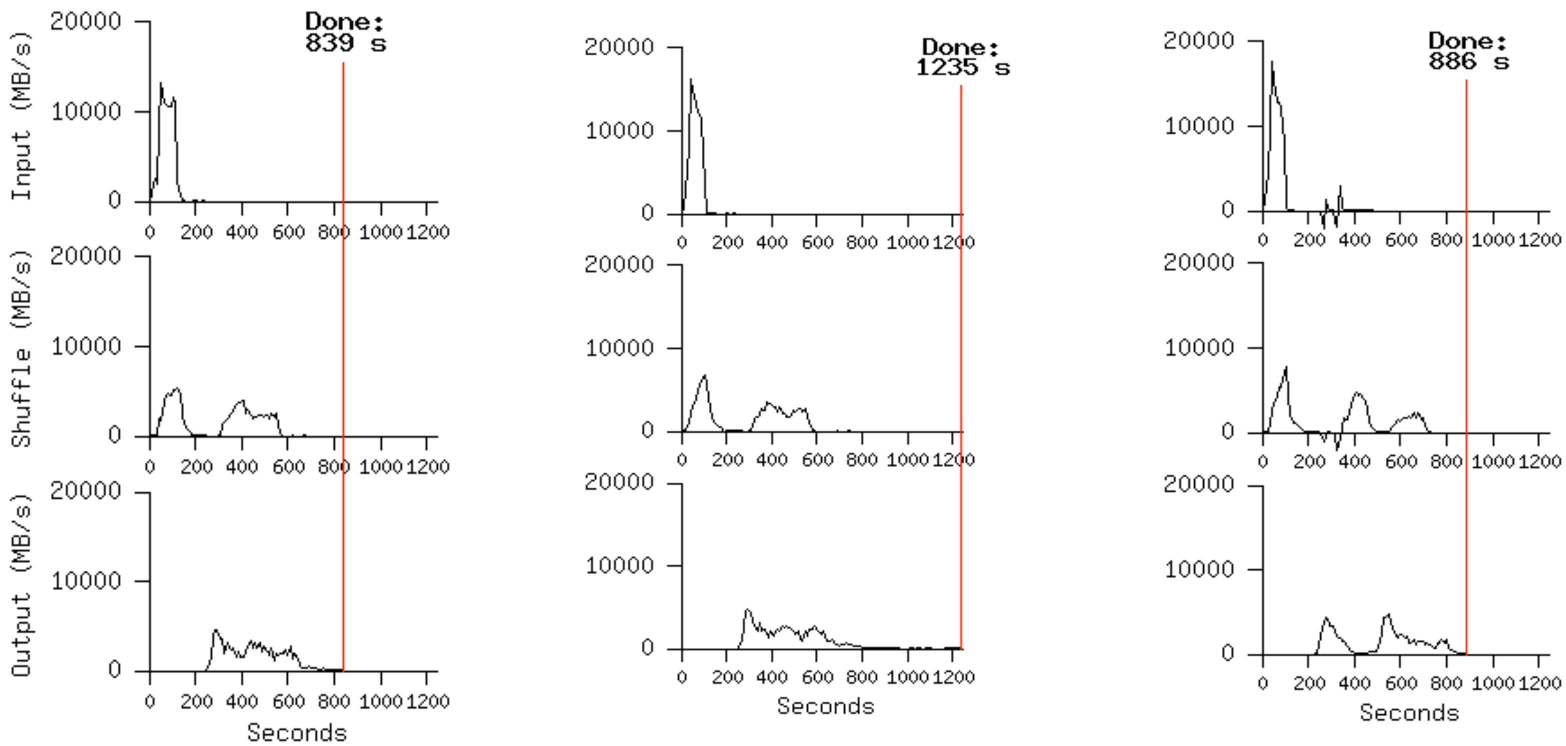
MR_Sort



Normal
No backup tasks
200 processes killed
Input (MB/s)
Shuffle (MB/s)
Output (MB/s)
20000
10000
0
0 200 400 600 800 1000 1200
Seconds
Done: 839 s
Done: 1235 s
Done: 886 s

MR_Sort

Normal | No backup tasks | 200 processes killed

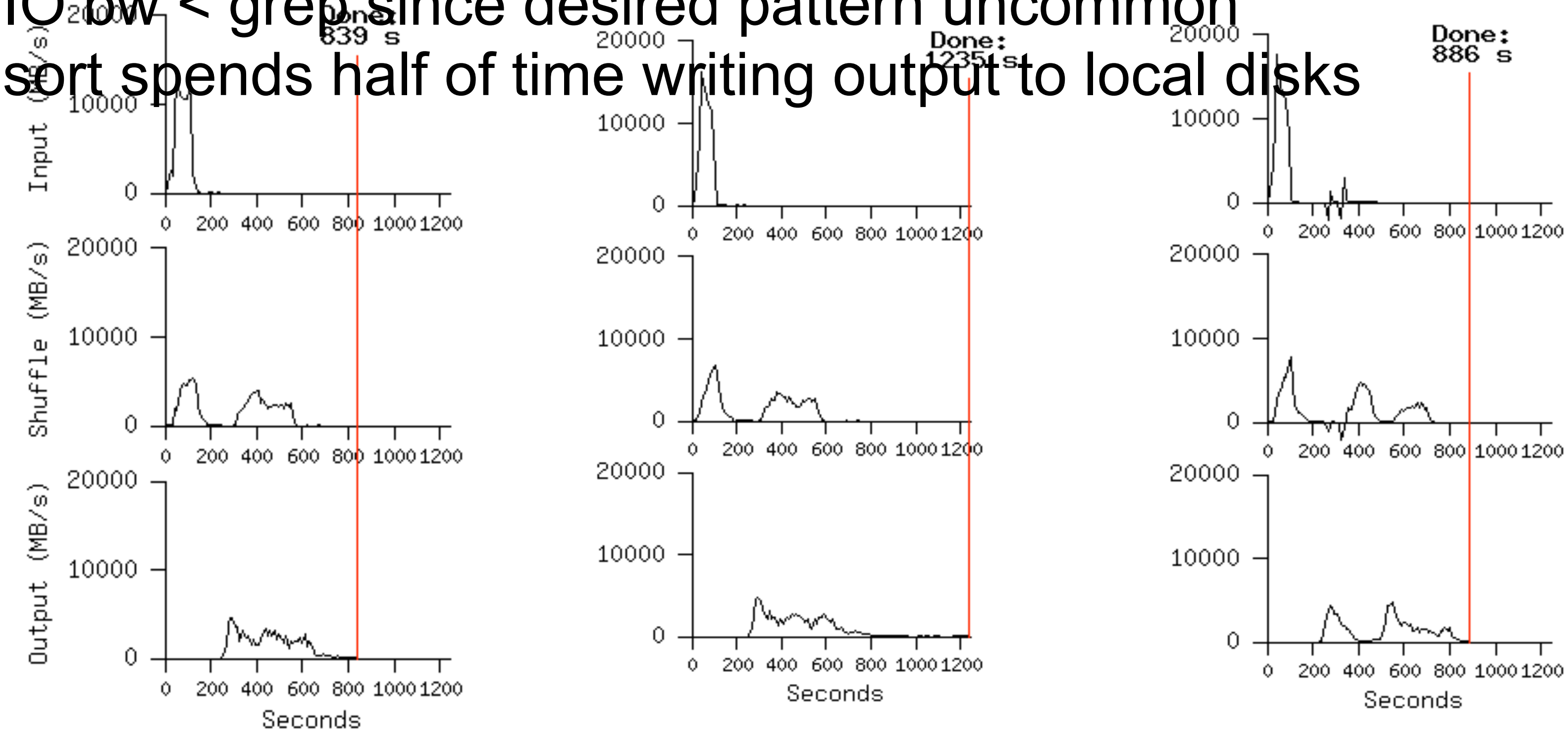


- Backup tasks reduce job completion time a lot!
- System deals well with failures

MR_Sort

Normal | No backup tasks | 200 processes killed

IO bw < grep since desired pattern uncommon
sort spends half of time writing output to local disks

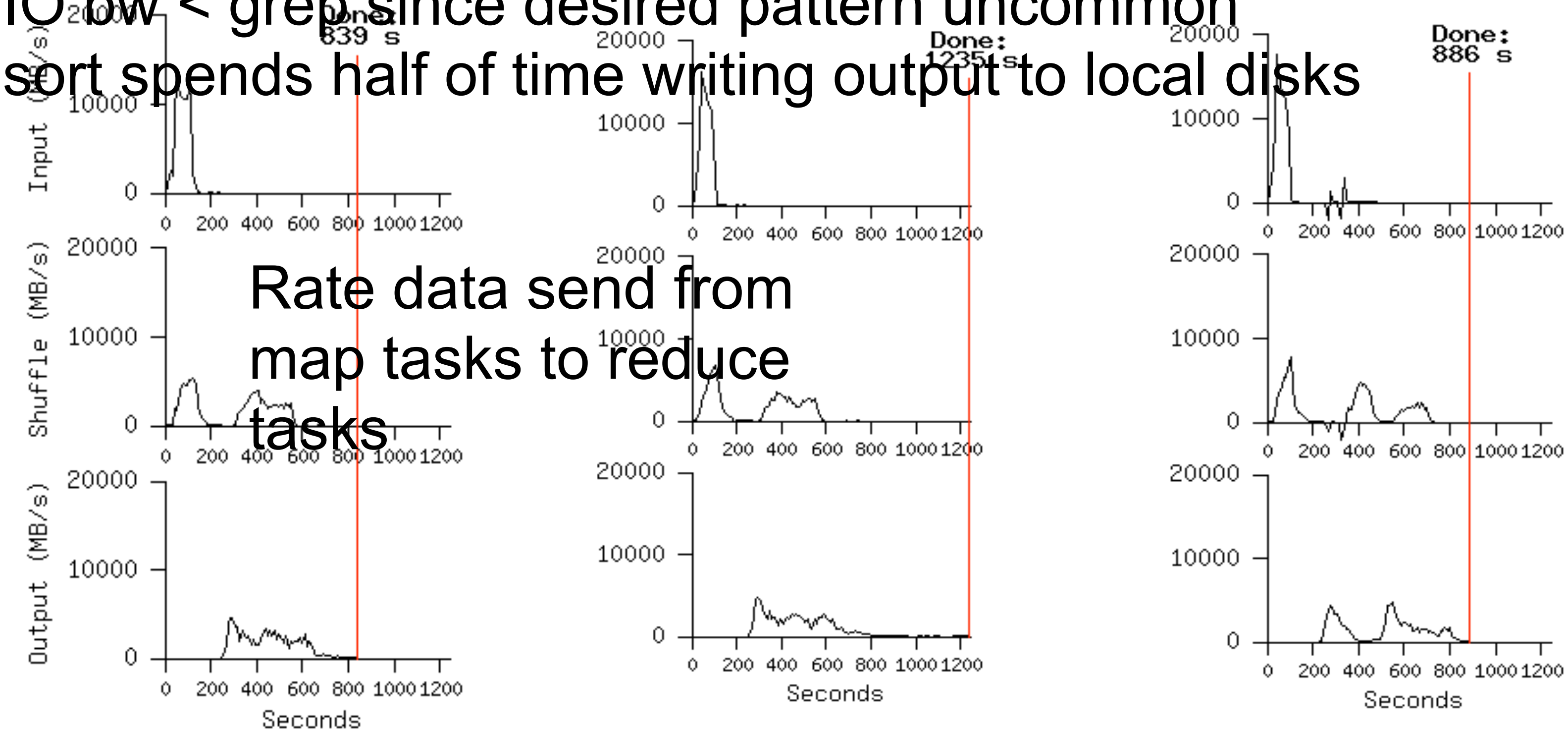


- Backup tasks reduce job completion time a lot!
- System deals well with failures

MR_Sort

Normal | No backup tasks | 200 processes killed

IO bw < grep since desired pattern uncommon
sort spends half of time writing output to local disks



Rate data send from map tasks to reduce tasks

- Backup tasks reduce job completion time a lot!
- System deals well with failures

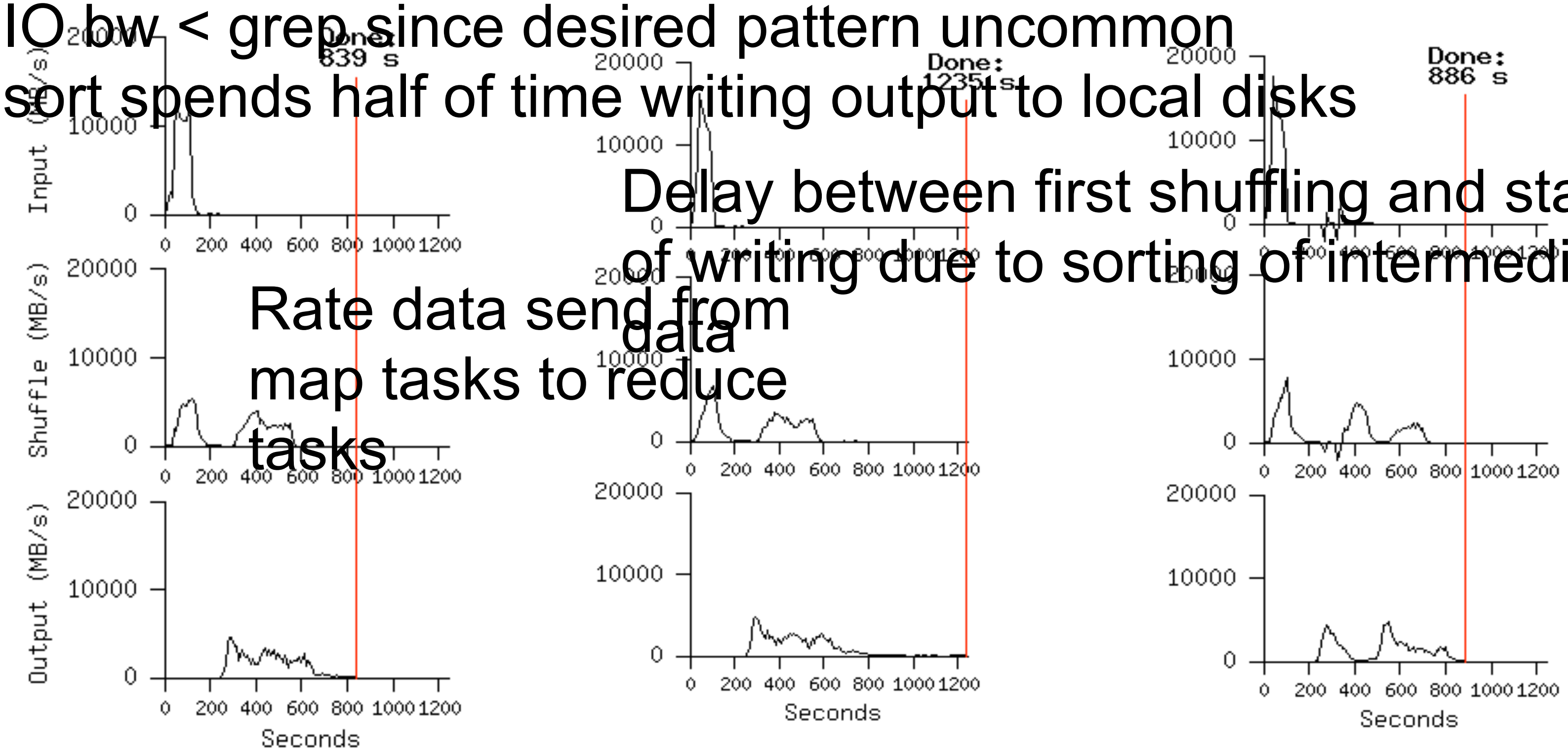
MR_Sort

Normal | No backup tasks | 200 processes killed

IO bw < grep since desired pattern uncommon
sort spends half of time writing output to local disks

Delay between first shuffling and start of writing due to sorting of intermediate data

Rate data send from map tasks to reduce tasks



- Backup tasks reduce job completion time a lot!
- System deals well with failures

MR_Sort

Normal | No backup tasks | 200 processes killed

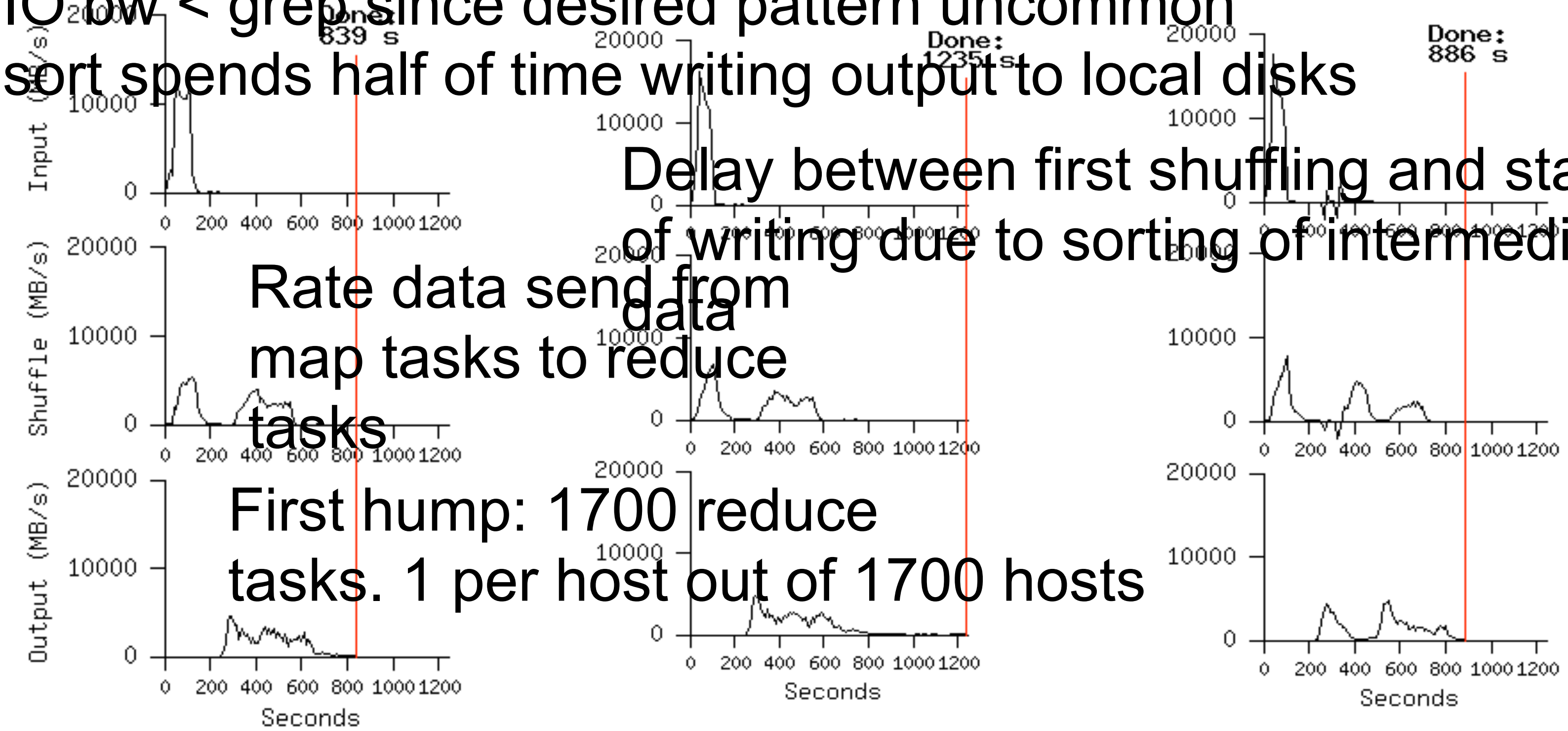
IO bw < grep since desired pattern uncommon

sort spends half of time writing output to local disks

Delay between first shuffling and start of writing due to sorting of intermediate data

Rate data send from map tasks to reduce tasks

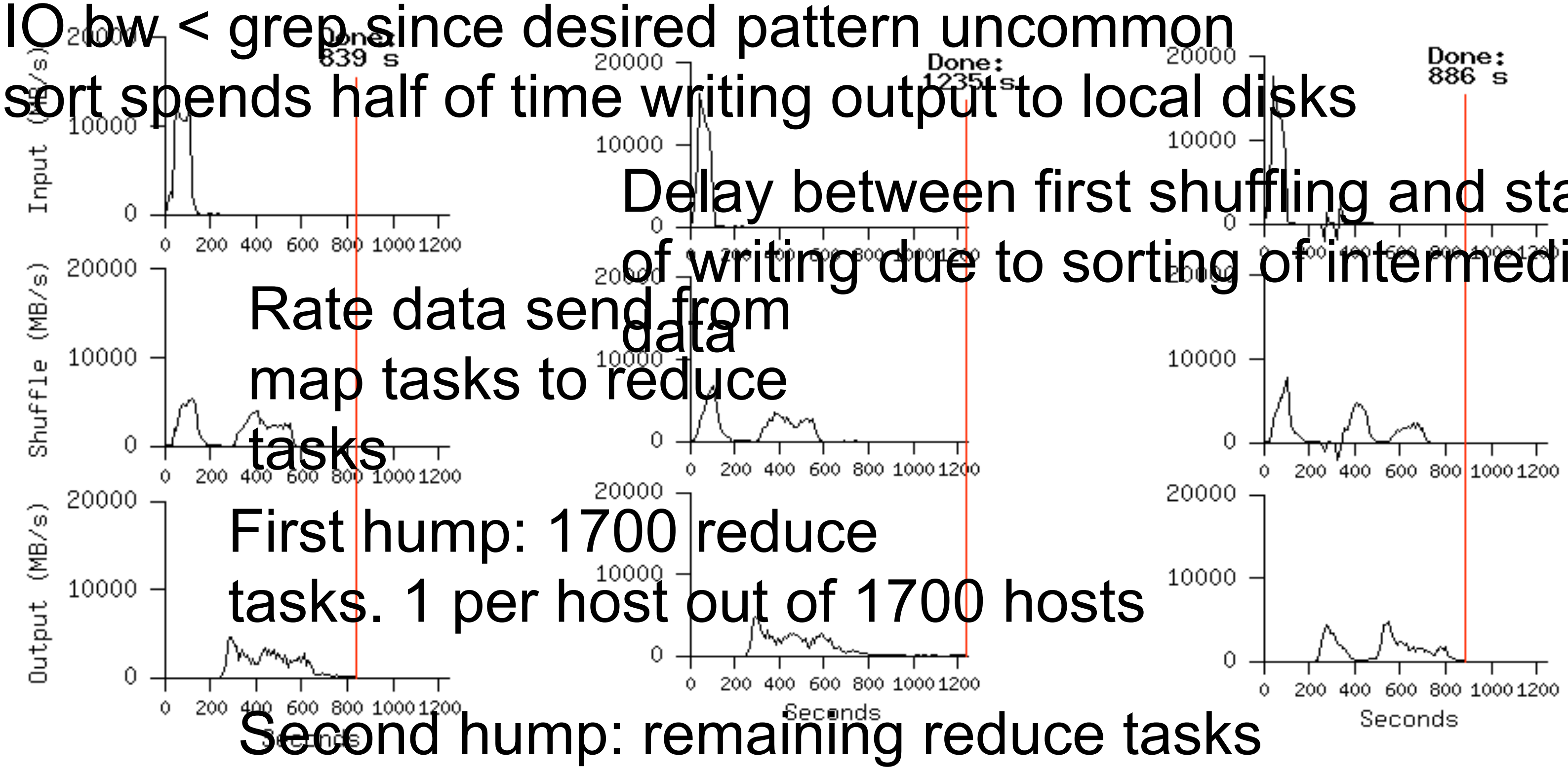
First hump: 1700 reduce tasks. 1 per host out of 1700 hosts



- Backup tasks reduce job completion time a lot!
- System deals well with failures

MR_Sort

Normal | No backup tasks | 200 processes killed



IO bw < grep since desired pattern uncommon
sort spends half of time writing output to local disks

Delay between first shuffling and start of writing due to sorting of intermediate data

Rate data send from map tasks to reduce tasks

First hump: 1700 reduce tasks. 1 per host out of 1700 hosts

Second hump: remaining reduce tasks

- Backup tasks reduce job completion time a lot!
- System deals well with failures

Usage in Aug 2004

Usage in Aug 2004

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days

Usage in Aug 2004

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB

Usage in Aug 2004

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2

Usage in Aug 2004

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2
Average map tasks per job	3,351

Usage in Aug 2004

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2
Average map tasks per job	3,351
Average reduce tasks per job	55

Usage in Aug 2004

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2
Average map tasks per job	3,351
Average reduce tasks per job	55

Conclusions

- MapReduce proven to be useful abstraction
- Greatly simplifies large-scale computations
- Fun to use:
  - focus on problem,
  - let library deal w/ messy details

A major step backwards

- http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html
- A giant step backward in the programming paradigm for large-scale data intensive applications
- A sub-optimal implementation, in that it uses brute force instead of indexing (hash / B-trees)
- Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
- Missing most of the features that are routinely included in current DBMS
- Incompatible with all of the tools DBMS users have come to depend on

Desktop Grids

- Use free compute, storage and network resources in Internet and Intranet environments
  - Reuse existing (power, resource)infrastructure
- Motivation
  - High return on investment
    - Savings often a factor 5 or 10 compared to dedicated cluster
  - Access to huge computational power and storage resources

State of the Art

- 400 TeraFlops/sec, over one million hosts

SETI@HOME   climateprediction.net   PREDICTOR @ home powered by BOINC

Loosely-coupled,
single application,
without time constraints

Tightly-coupled,
multiple applications,
with time constraints

State of the art

Challenges

- Volatility
  - Resources are shared
    - Mouse/keyboard activity, user processes
  - Nondeterministic failures
    - Often 50% failure rates
- Heterogeneity
- Accessibility
  - Resources are behind NAT's, firewalls
- Security

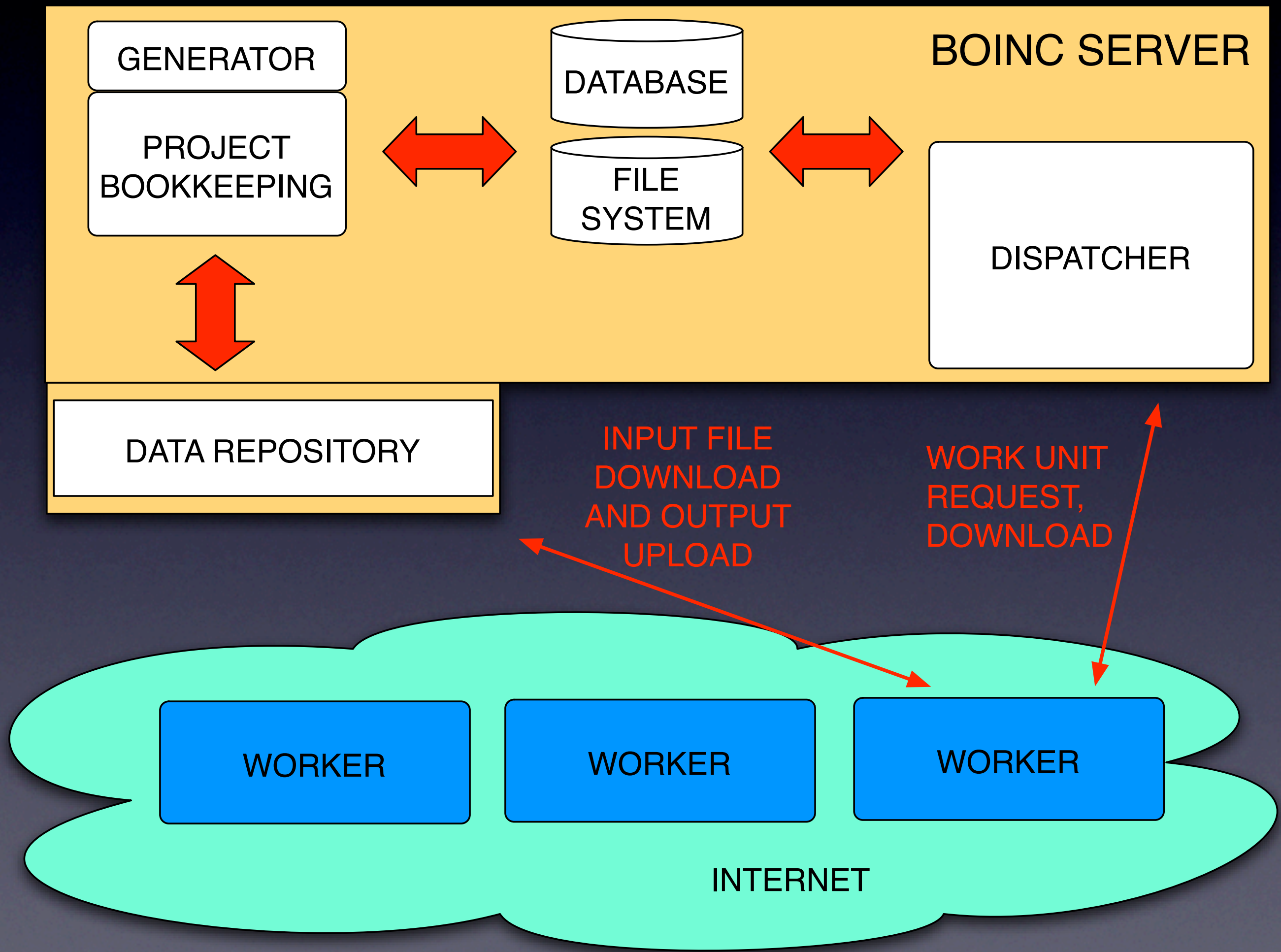
Outline

- BOINC
- XtremWeb
- Prediction

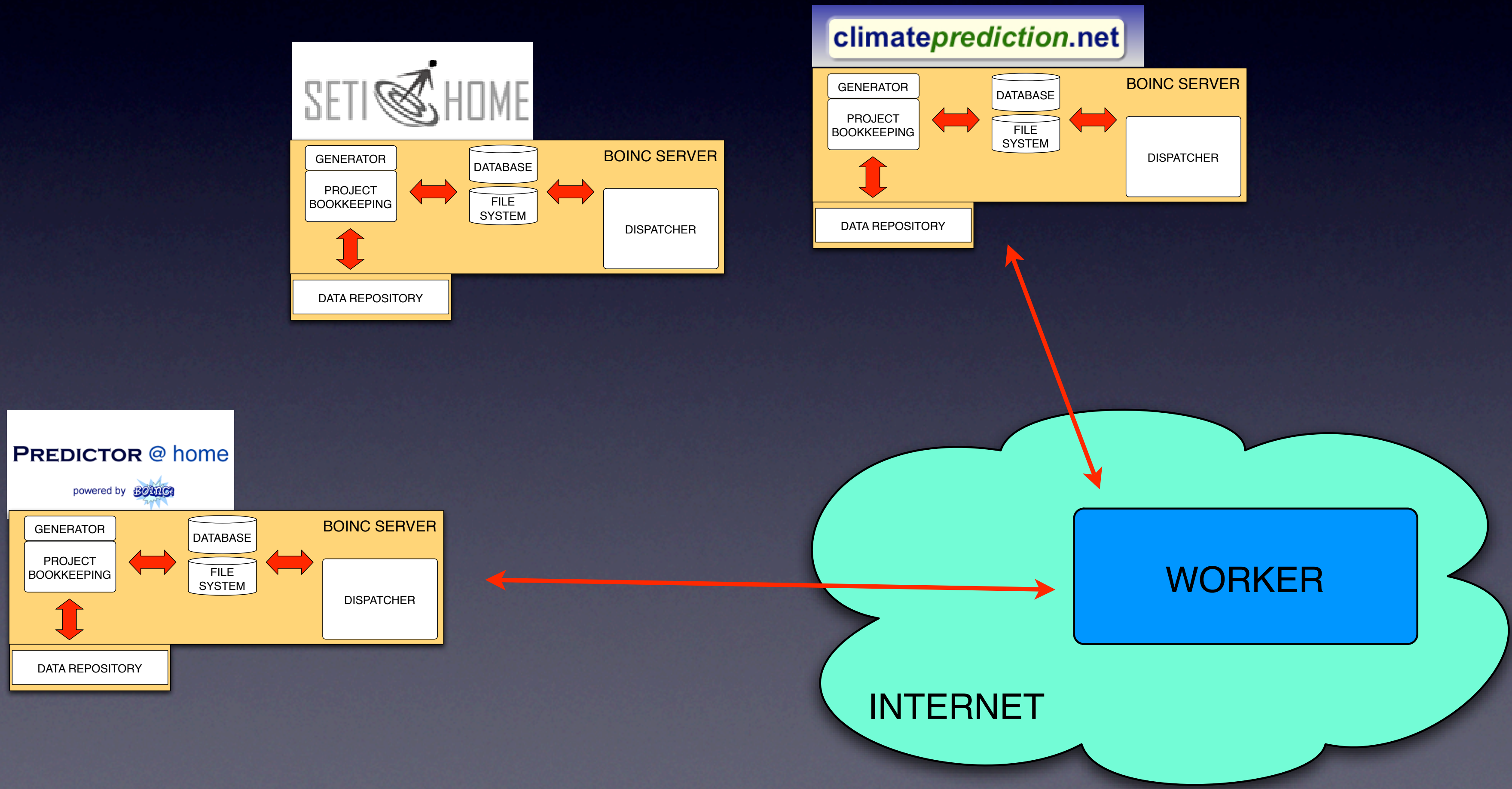
BOINC

- Background
  - Led by David Anderson, UC Berkeley
  - SETI@home
  - Single astronomy application
  - Too many resources
- Goals of BOINC
  - Ability to share resources among multiple projects
  - User autonomy
  - Usability

BOINC Architecture



BOINC Architecture



BOINC Worker Scheduling Problem

- Workers have resource share (CPU) allocation per project
- Work units per project have a deadline
- Goal: meet deadline and also resource share allocations
- Which project to schedule next on worker?

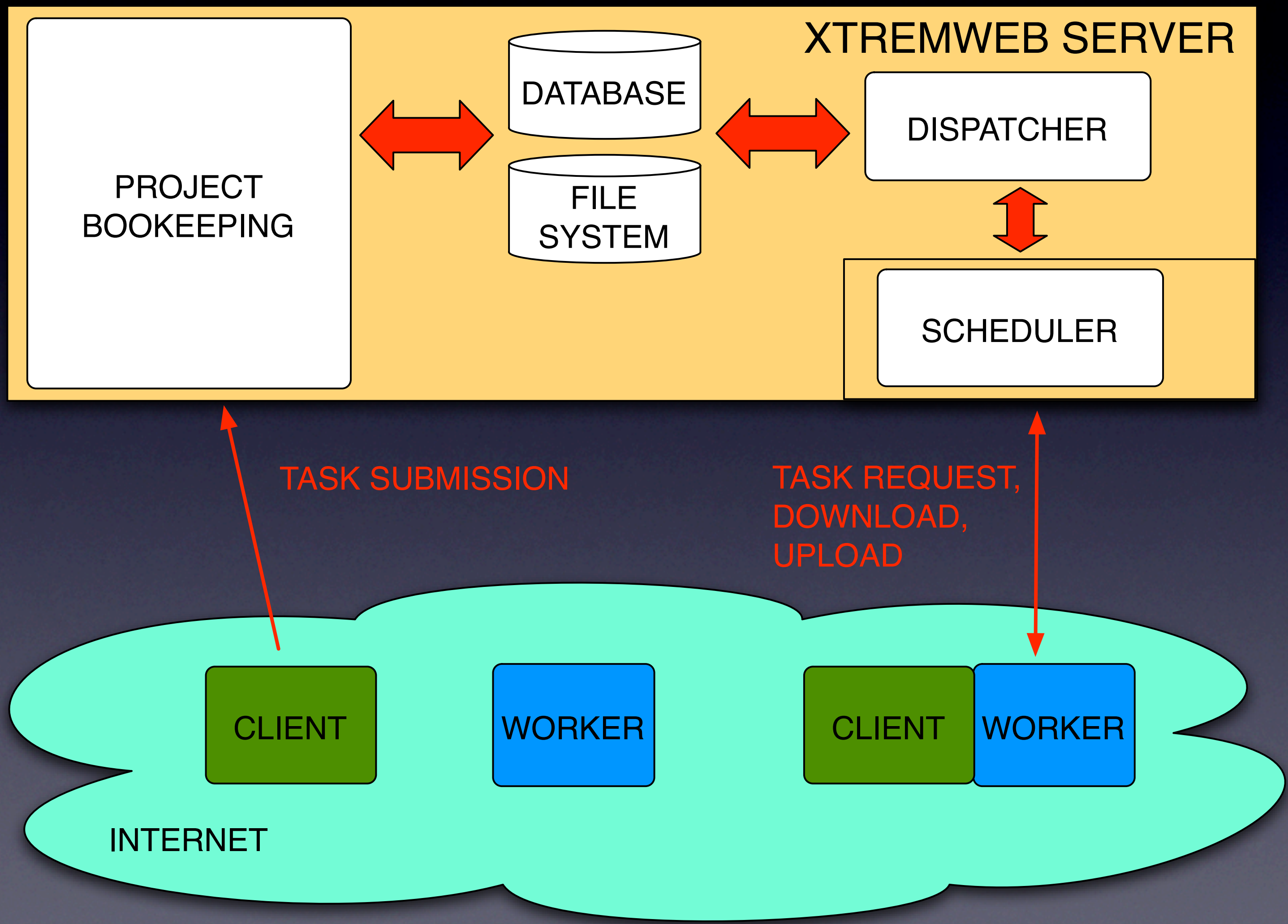
BOINC Scheduling Approach

- Use weighted round robin until a project risks missing deadline
  - If so, switch to earliest deadline first scheduling
- N.B.: scheduling depends on many different parameters (e.g., availability of the resources, resource hardware, user preferences, task deadlines, resource shares, estimates of task completion time, number and characteristics of projects)

XtremWeb

- Led by Gilles Fedak (fedak@lri.fr), INRIA Futurs
- Goals
  - Support symmetric needs of users
    - Allow any node to play any role (client, worker)
  - Fault tolerance
  - Usability

XtremWeb Architecture



Ensuring Collective Availability in Volatile Resource Pools via Forecasting

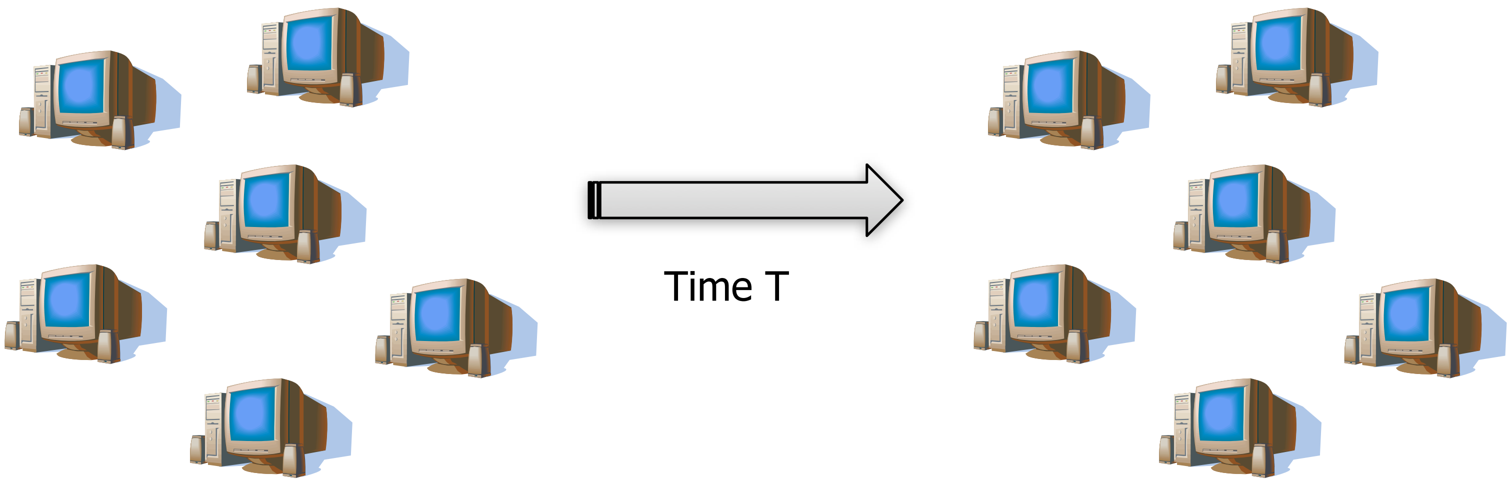
Artur Andrzejak	Zuse-Institute Berlin (ZIB)
Derrick Kondo	INRIA
David P. Anderson	UC Berkeley

Motivation

- Goal: can we deploy serious services / apps over unreliable resources?
- How unreliable?
  - mostly non-dedicated PC's (used for other purposes)
  - e.g. volunteer computing Grids such as SETI@home
  - no control over availability, frequent churn
- What are "serious" services / apps?
  - large scale service deployment
    - examples: Amazon's EC2, TeraGrid, EGEE
  - complex applications
    - examples: DAG/message-passing applications
  - high availability: around 99.999

How to do this?

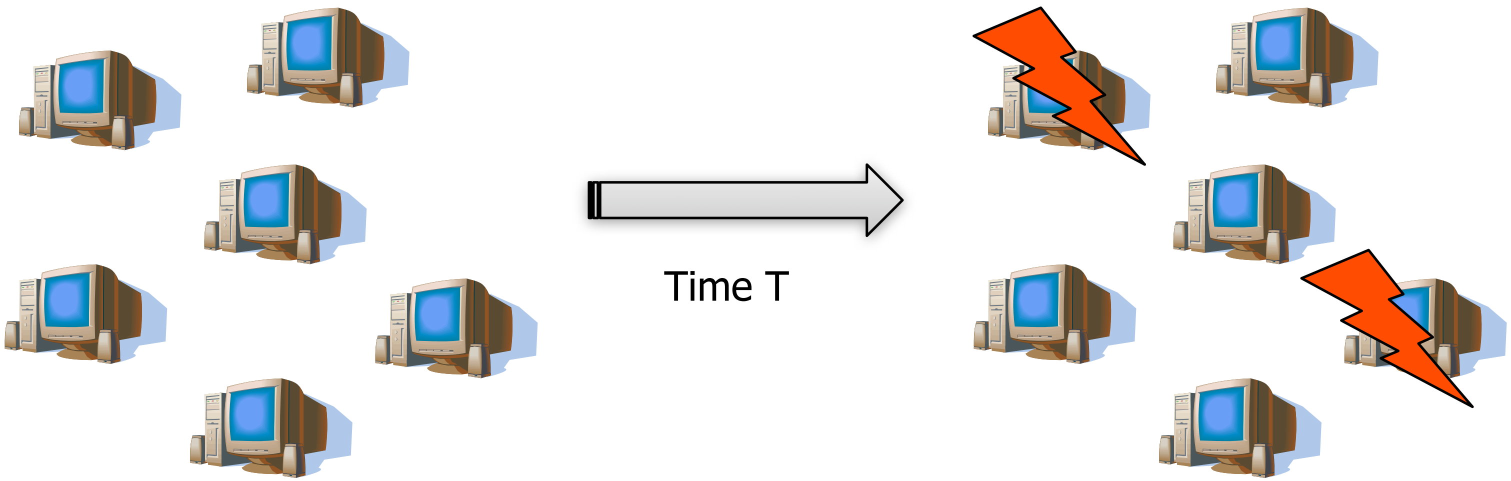
- Difficult to get (many) hosts with high avail
- Instead, we strive for collective availability:
  - def.: *guarantee that with high probability, in a group of $R \geq N$ hosts, at least N remain available over time T*



R = 6, N = 3

How to do this?

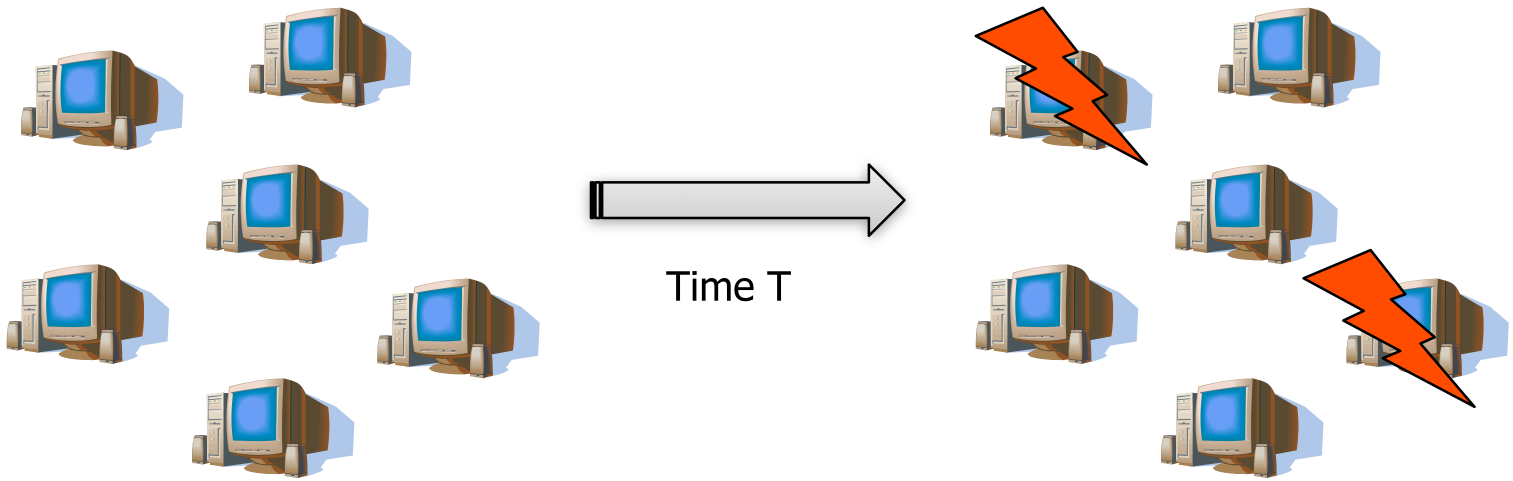
- Difficult to get (many) hosts with high avail
- Instead, we strive for collective availability:
  - def.: *guarantee that with high probability, in a group of $R \geq N$ hosts, at least N remain available over time T*



R = 6, N = 3

How to do this?

- Difficult to get (many) hosts with high avail
- Instead, we strive for collective availability:
  - def.: *guarantee that with high probability, in a group of $R \geq N$ hosts, at least N remain available over time T*

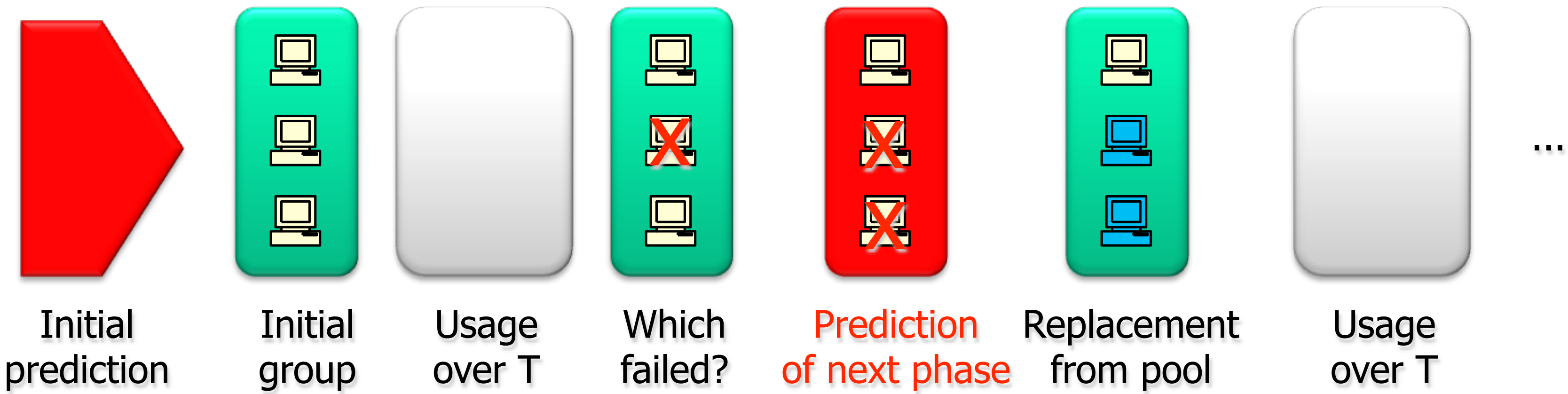


R = 6, N = 3

4 ≥ N survived, col. availability achieved

Our Focus

- We use statistical and prediction methods to answer the question:
  - *Given a pool of non-dedicated hosts and a request for N hosts, how to select them such that the collective availability is maximized?*
    - i.e. at least N among R hosts "survive" interval T
- Then deployment:



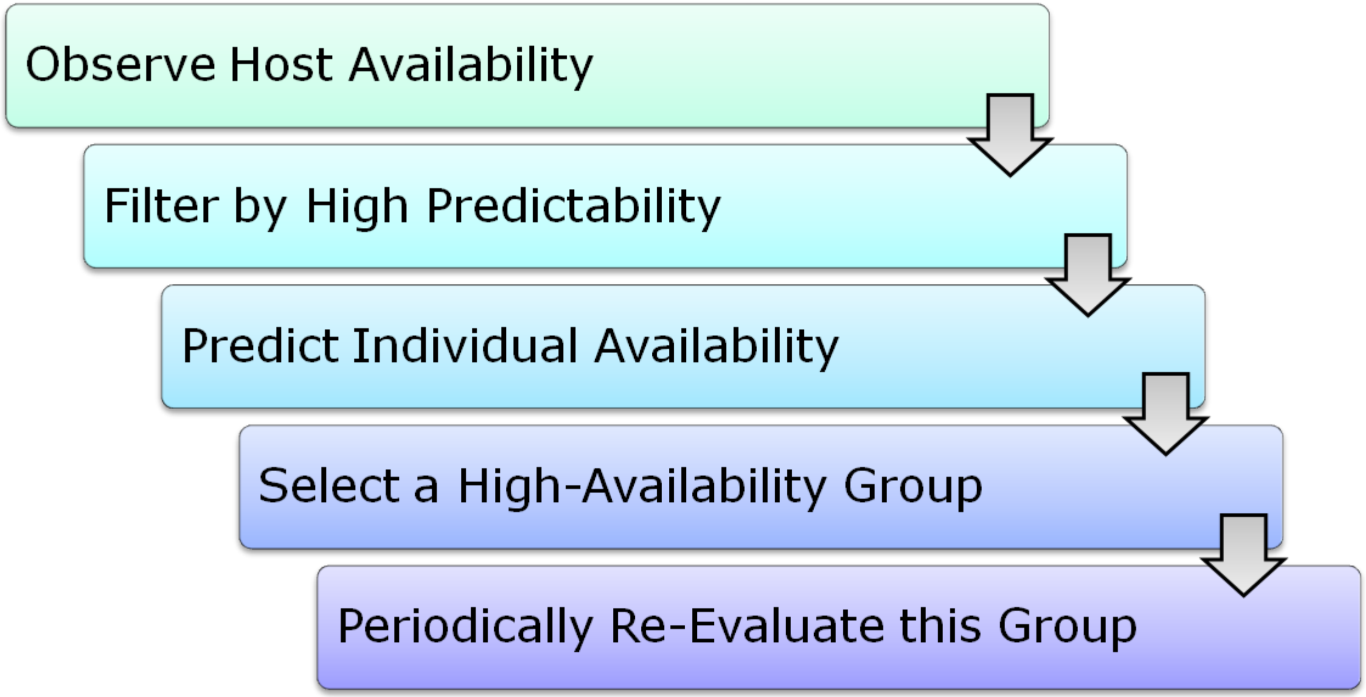
Availability Prediction

- We propose efficient and domain-adjusted predictions of availability for individual hosts
  - efficient:
    - fast pre-selection of predictable hosts
    - use simple and fast classification algorithm
  - domain-adjusted
    - analyze the factors of predictability and adjust our methods to them
- Then we use these individual predictions to achieve collective availability

Measurement Data

- Availability traces for over 48,000 hosts participating in SETI@home
- Active in Dec 1st, 2007 to Feb 12th, 2008
- Availability recorded by a BOINC client
  - depends whether the machine was idle
  - The definition of idle depends on user settings
- Quantized to 1 hour intervals
  - regarded as available only if *uninterrupted* avail for the *whole* hour – quite conservative
- For availability characterization, see:
  - Derrick Kondo, Artur Andrzejak, David P. Anderson: ***On Correlated Availability in Internet-Distributed Systems***, 9th IEEE/ACM International Conference on Grid Computing (Grid 2008), Tsukuba, Japan, September 29-October 1, 2008

Prediction Process



Filtering Hosts By Predictability

- We want to find out, for each host, whether its availability predictions are likely to be accurate
- I.e. we want hosts with high predictability:
  - def.: *expected accuracy of predictions from a model build on historical data*
- To estimate it, we use indicators of predictability
  - fast to compute (at least faster than a prediction model)
  - use only training data

Predictability Computation

- To assess the accuracy of predictability indicators, we have to compute for each host the true accuracy of model-based predictions
- To this end, we train a prediction model on the historical availability data (4 weeks @ 1 hour), and then compute the prediction error on the subsequent 2 weeks (1 hour => 2*7*24 predictions)
  - This is only the "laboratory" scenario, not done in real deployment
  - The predictability indicators should tell us, for which hosts it is not worth to build model / do predictions

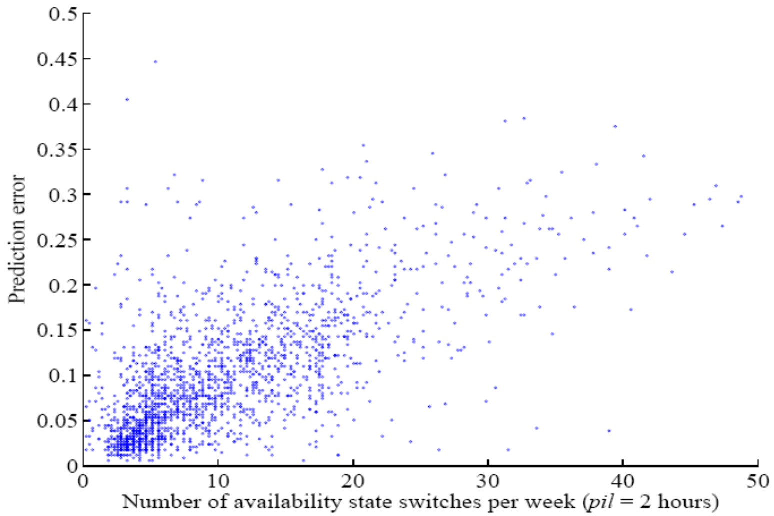
Predictability Indicators

- We have tested, among others:
  - Average length of an uninterrupted availability segment
  - Size of the compressed availability trace
    - traces with predictable patterns are likely to compress better
  - Prediction error tested on a part of the training data (as a "control indicator")
  - Number of availability state changes per week (aveSwitches)
- Evaluation:
  - correlation, scatter plots

And the winner is..

- Number of availability state changes per week: aveSwitches

Scatter plot
aveSwitches vs.
prediction error



Spearman's rank
correlation coefficient
indicator vs.
prediction error

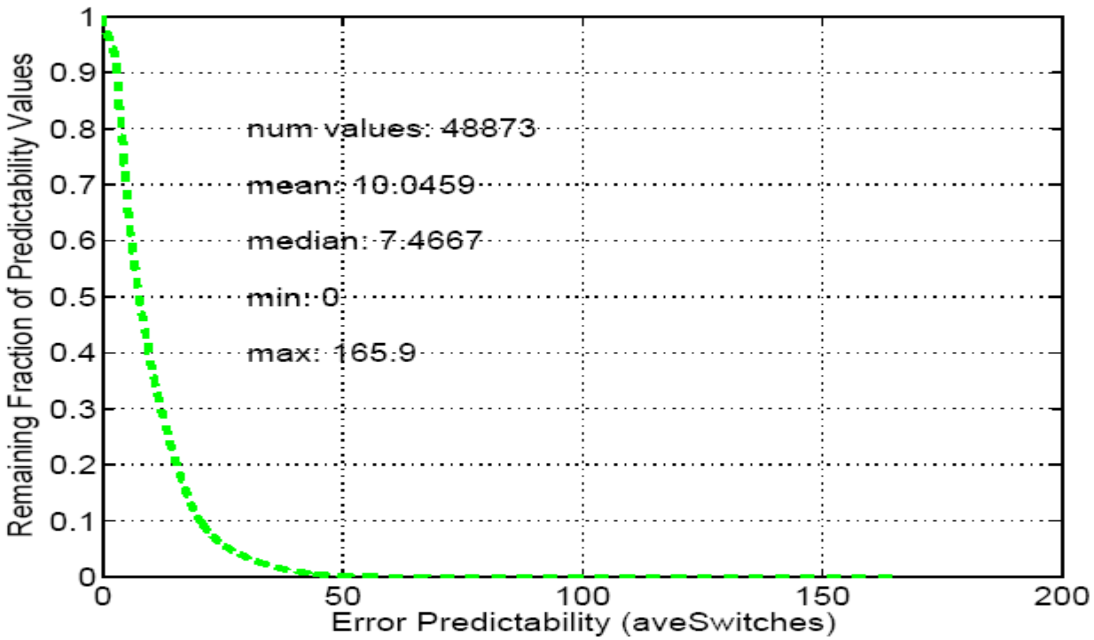
pil	$aveAva$	$aveAvaRun$	$aveNavaRun$	$aveSwitches$	$zipPred$	$modelPred$
1	-0.370	-0.594	0.085	0.707	-0.654	-0.724
2	-0.275	-0.486	-0.011	0.678	-0.632	-0.690
4	-0.119	-0.303	-0.119	0.548	-0.502	-0.640
8	0.194	0.056	-0.245	0.195	-0.127	-0.642
16	0.211	0.091	-0.185	0.057	0.062	-0.568

Why are AveSwitches good?

- There are some "reasons" for data regularity → high prediction accuracy
  1. Periodic behavior, e.g. daily periodicities
  2. Long runs of availability / non-availability
  3. …
- We have studied which "reasons" are dominant:
  - by using data preprocessing which "helps" either 1 or 2
- results show that "reason" 2 is dominant
- highest accuracy for a mixture of both "reasons"

Filtering by predictability

- We create two groups for further processing:
  - low predictability, with aveSwitches ≥ 7.47
  - high predictability, with aveSwitches < 7.47



Prediction Background

- Def. of classifier: a function which learns its output value from examples
- Function inputs are called attributes, in our study:
  - Functions of availability represented as 01 binary string
    - Time (e.g. hour in day), history bits (sum of recent k history bits)
- Output is an element from some fixed set, in our study:
    - {0,1} representing availability

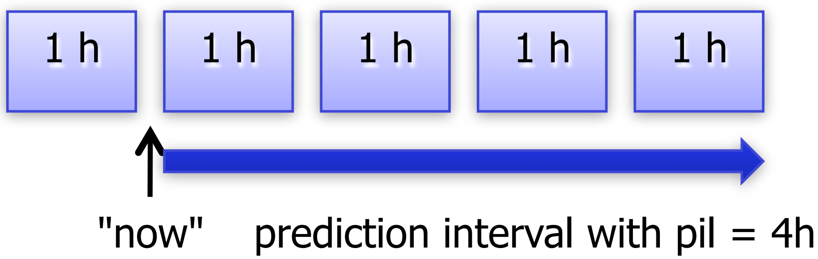
	Attribute_1	…	Attribute_n	**Output**
Example 1	[23,10]	…	[21,5]	0
Example k	[11,10]	…	[5,0]	1
Prediction	[1,7]	…	[13,7]	**?**

learn

predict

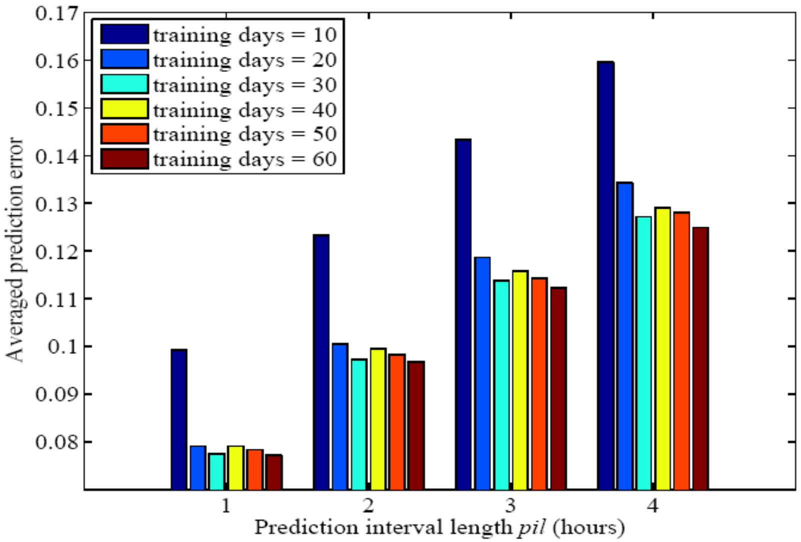
Prediction

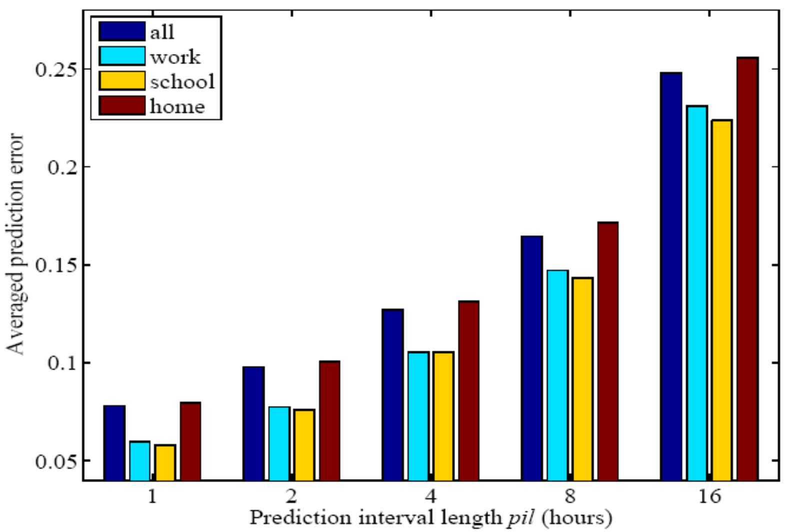
- We have used a simple and fast classifier
  - naive Bayes
- The classifier takes examples i.e. vectors of measured avail + preprocessed data over 30 days
- Predicts for each hour over two weeks
  - starting now, will the host be available in the next k hours
    - this is prediction interval length, pil



What drives accuracy?

- Dependence upon
  - prediction interval length, pil
  - training interval length
  - host ownership type (private, school, work)



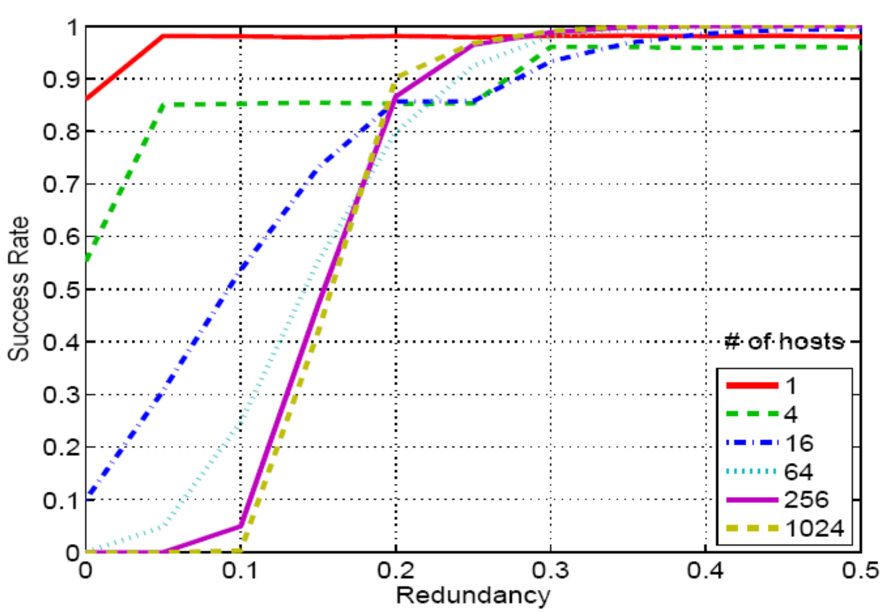


Simulation Approach

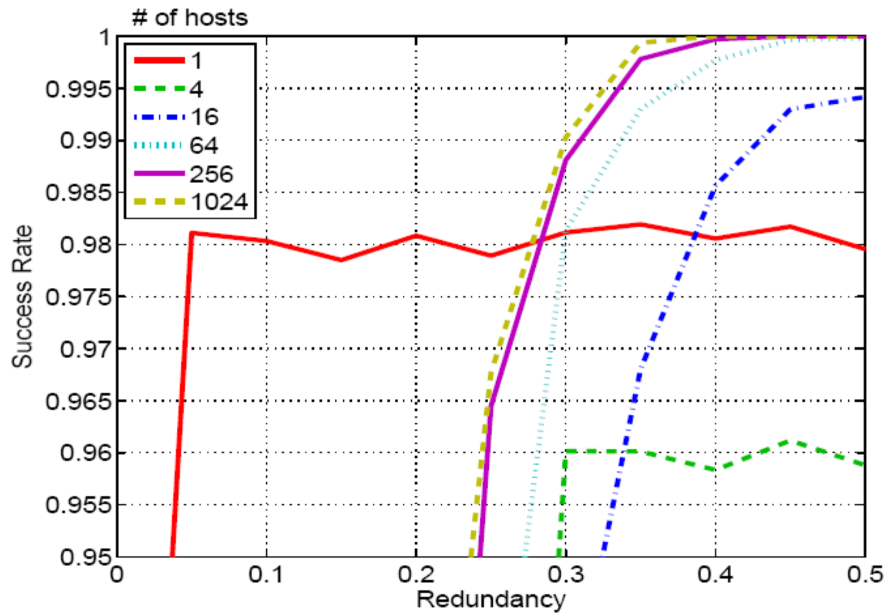
- For each host in the high-predictability group make prediction at t_0 for pil time, and select random R among those predicted as availiable
- R depends upon:
  - N = the desired number of hosts (at least N should be always available)
  - the redundancy (R-N)/N
- Our simulations answer:
  - given N and α, the desired availability level, what is the necessary redundancy, i.e. necessary R?
  - a little weaker: success rate: ratio (# experiments with at least N hosts alive after time T) / (all experiments)

Necessary Redundancy

- High predictability group (pil=4)



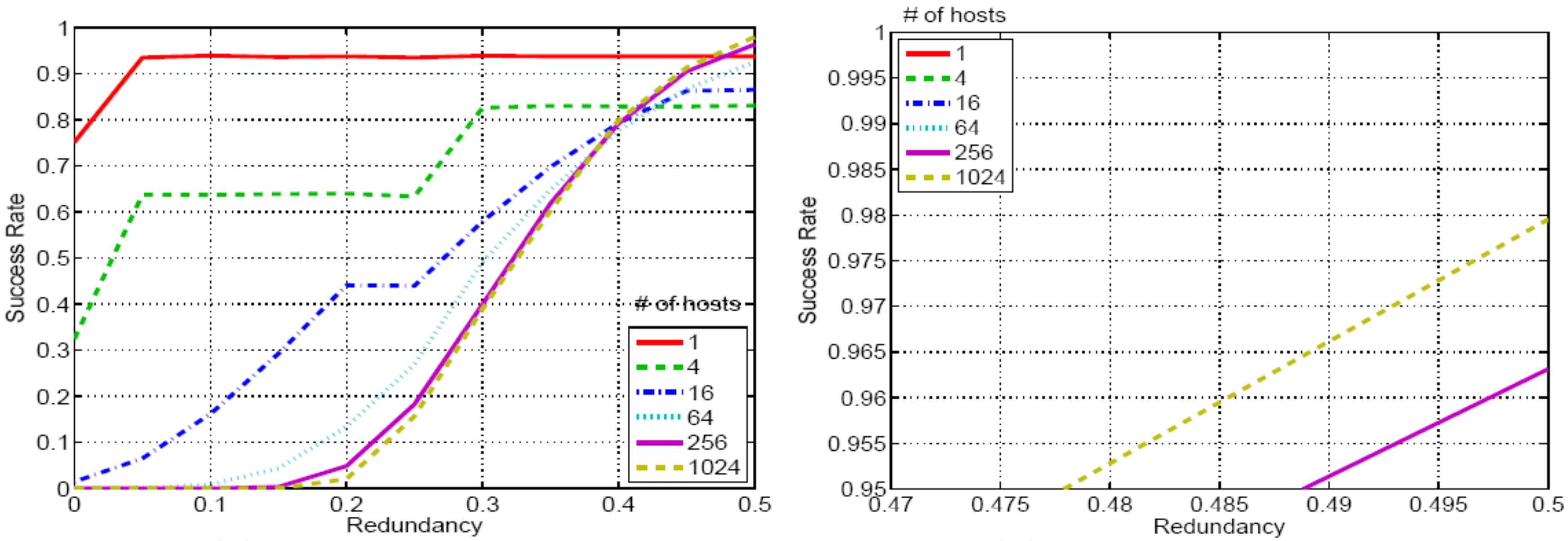
(a) Complete range



(b) Zoomed-in range

Necessary Redundancy

- Low predictability group (pil=4)



(a) Complete range (b) Zoomed-in range

Is this Redundancy too high?

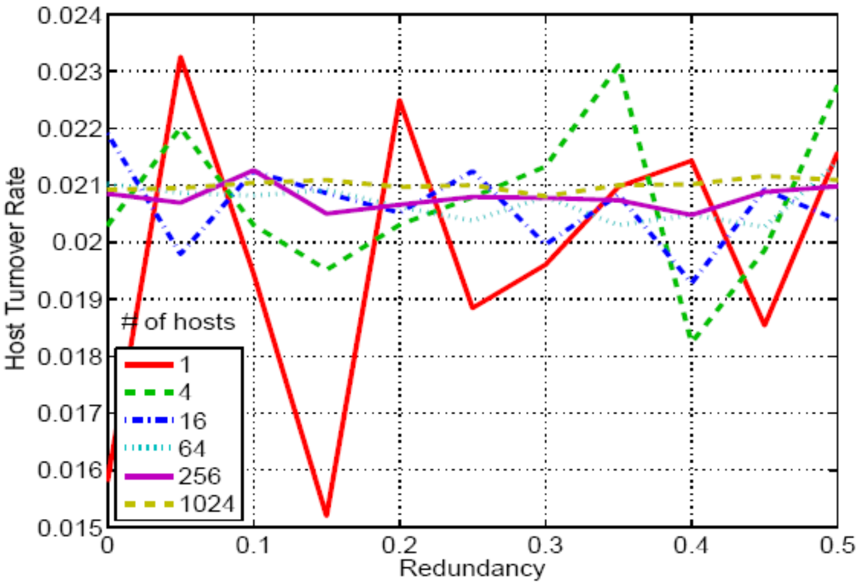
- In high predictability group, we have required redundancy of 35%
- However, we consider this dramatically low
  - In comparison, SETI@home has 200% redundancy (also used for result validation)
  - In terms of absolute savings, that equates to 165 TeraFLOPS saved in a 1 PetaFLOPS system (such as FOLDING@home) => significant power savings
- As a result, the BOINC consortium is interested in potentially applying our prediction schema in their job scheduling (preliminary talks)

Migration Overhead

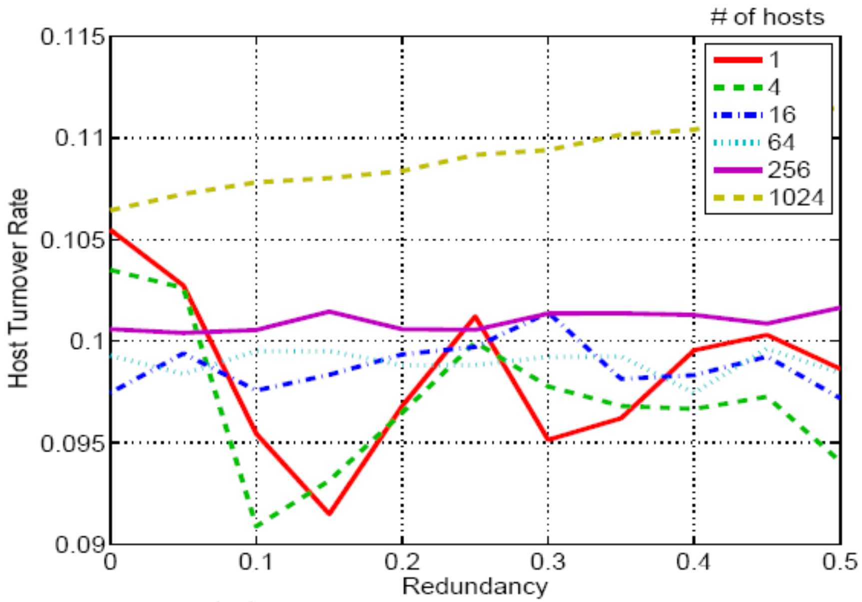
- We also evaluated the overhead due to host migration, service restart between slices of len T
- Threshold = a multiple of pil which describes the total time (many T's) of running an app / service
- Turnover rate TR:
  - let S be a set of hosts predicted to be available at t_0
  - for those we predict which ones become not available after time pil, i.e. second prediction at t_0+T
  - TR is the fraction of hosts which change from avail to non-avail
  - essentially, the higher, the more migration needed

Turnover Rates

- about 2.5% for high predictability group
- about 12% for low predictability group



(a) High predictability



(b) Low predictability

Summary

- Given that host redundancy is not an issue ("cheap" resources), high collective availability is achievable
  - even with low migration costs
- Predictability assessment and filtering is essential
  - improves accuracy
  - avoids many "wasted" predictions
- Future work:
  - hardest part: a new "application architecture" / programming model for collective availability
  - masking failures by virtualization and VM migration

References

- This work has been accepted at:
  - 19th IFIP/IEEE Conference on Distributed Systems: Operations and Management (DSOM 2008) (part of Manweek 2008), Samos Island, Greece, September 22-26, 2008
- Pdf available on request, please send e mail (derrick.kondo [at] inria.fr)

Reverse Web-Link Graph

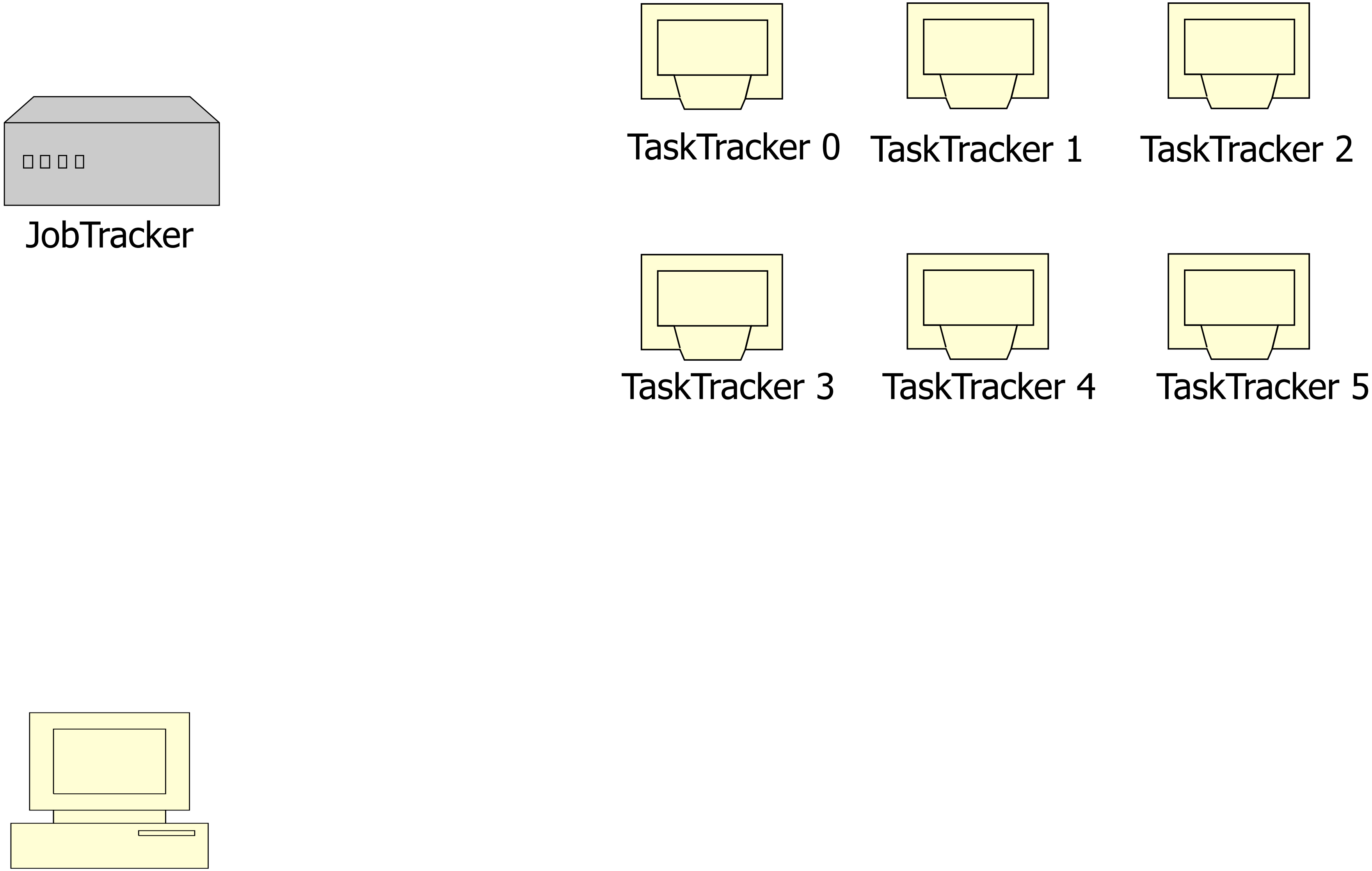
- Map
  - For each URL linking to target, …
  - Output <target, source> pairs
- Reduce
  - Concatenate list of all source URLs
  - Outputs: <target, **list** (source)> pairs

Inverted Index

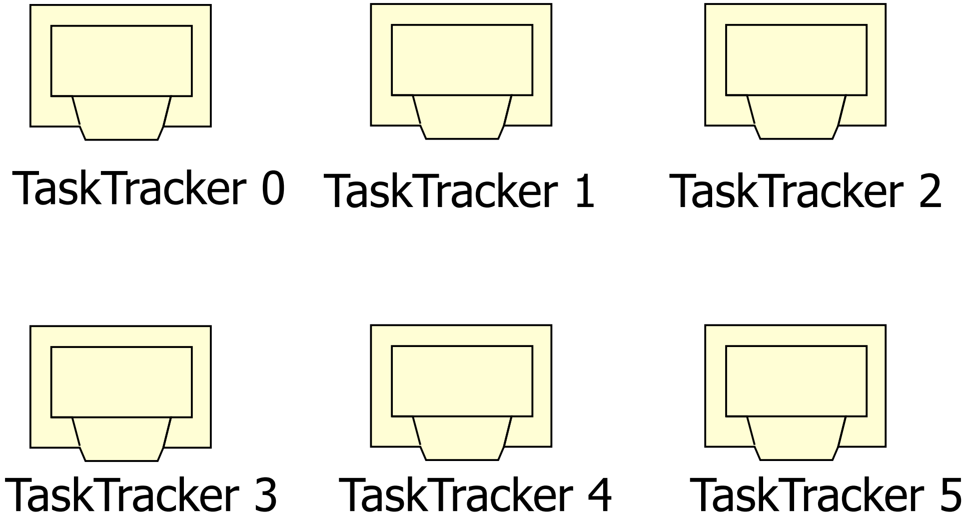
- Map ()
  - emit <word, document ID>

- Reduce
  - emit <word, list (document ID)>

Job Processing

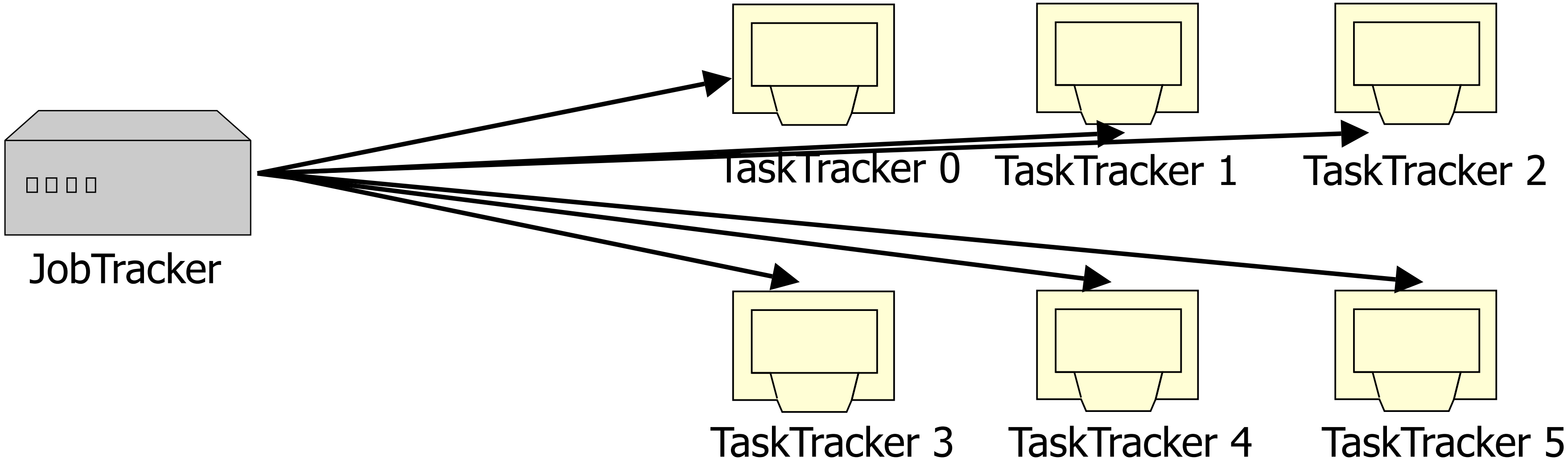


Job Processing

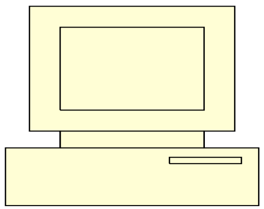


1. Client submits "grep" job, indicating code and input files

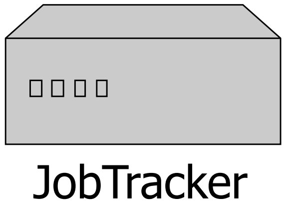
Job Processing

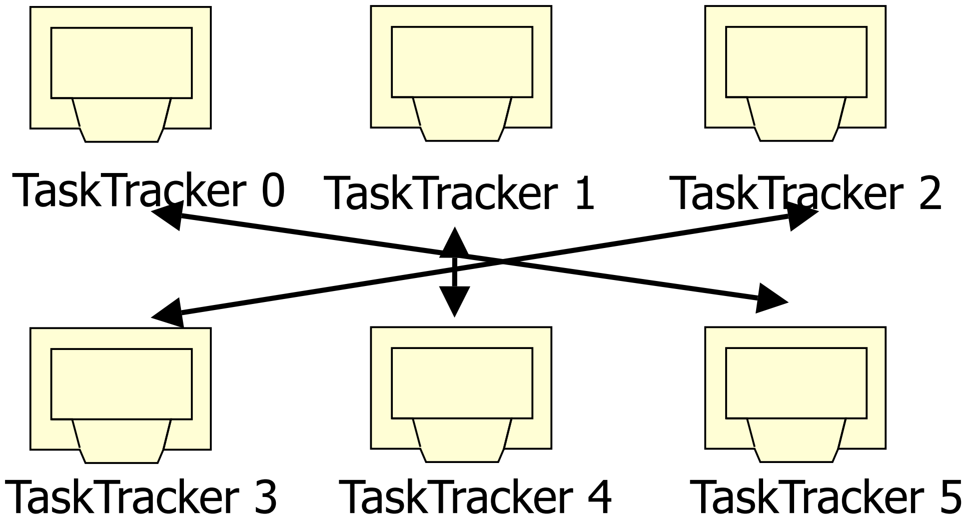


1. Client submits "grep" job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to ttrackers.

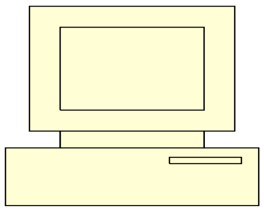


Job Processing

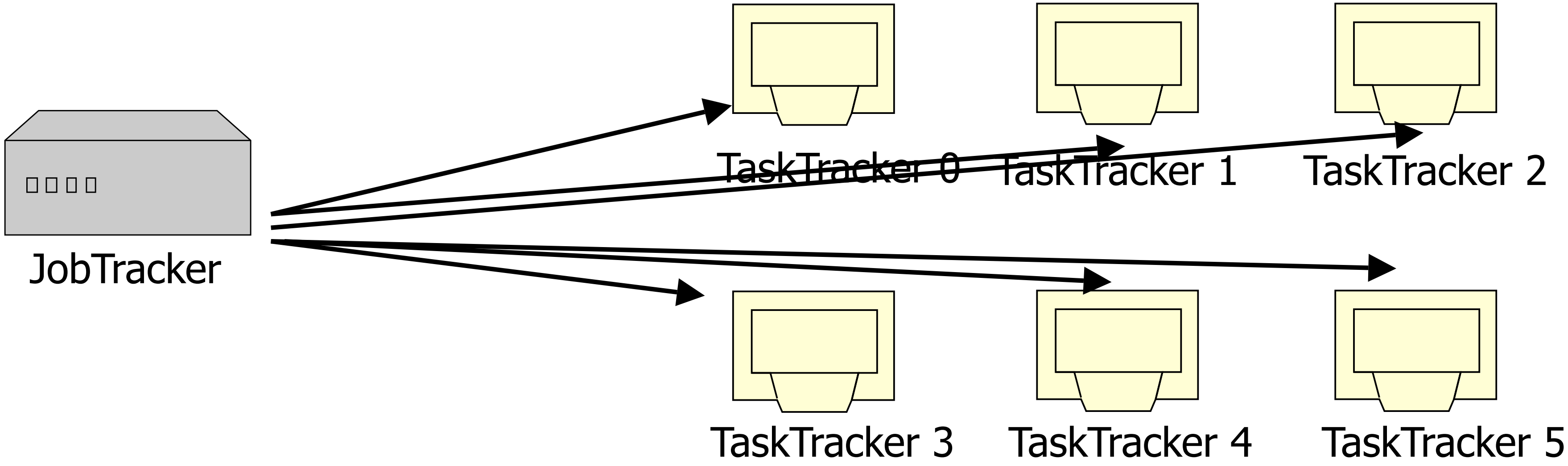




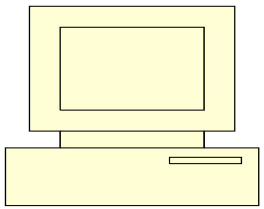
1. Client submits "grep" job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to ttrackers.
3. After map(), tasktrackers exchange map-output to build reduce() keyspace



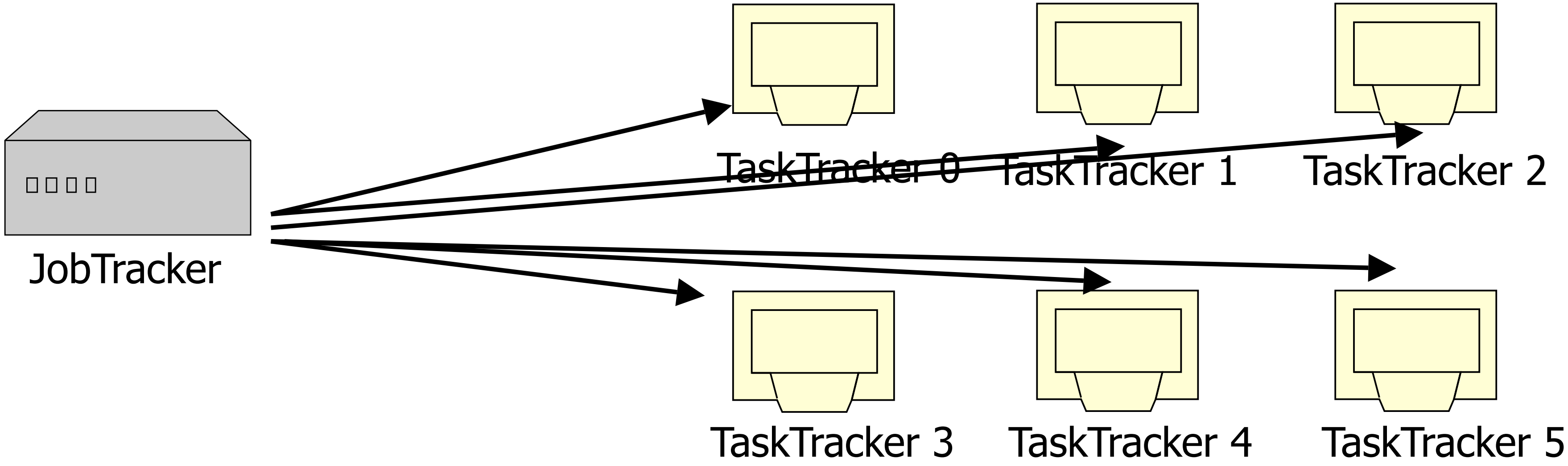
Job Processing



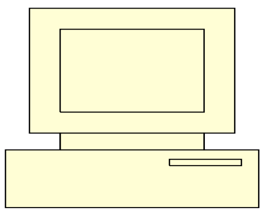
1. Client submits "grep" job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to ttrackers.
3. After map(), tasktrackers exchange map-output to build reduce() keyspace
4. JobTracker breaks reduce() keyspace into m chunks (in this case 6). Assigns work.



Job Processing

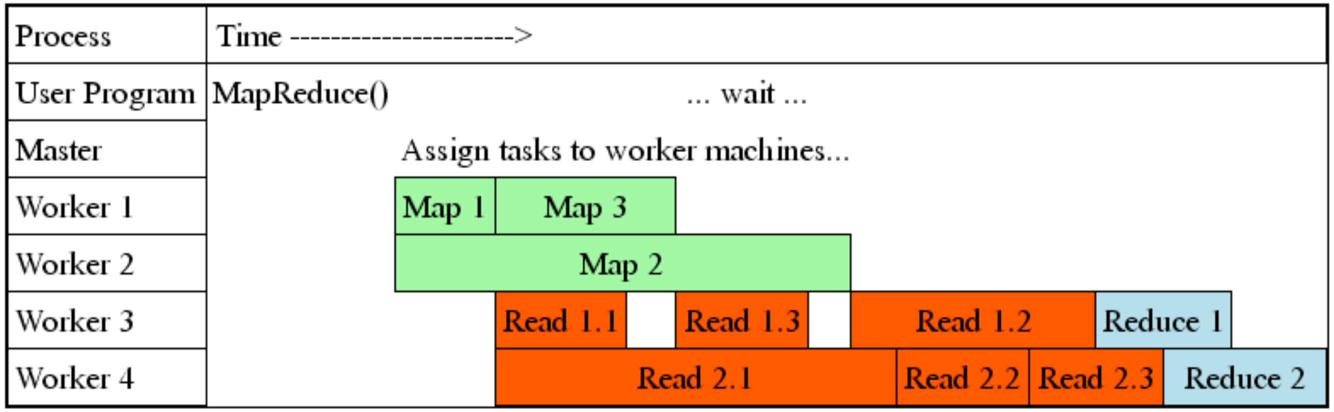


1. Client submits "grep" job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to ttrackers.
3. After map(), tasktrackers exchange map-output to build reduce() keyspace
4. JobTracker breaks reduce() keyspace into m chunks (in this case 6). Assigns work.
5. reduce() output may go to NDFS



Task Granularity & Pipelining

- Fine granularity tasks: map tasks >> machines
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing
- Often use 200,000 map & 5000 reduce tasks
- Running on 2000 machines

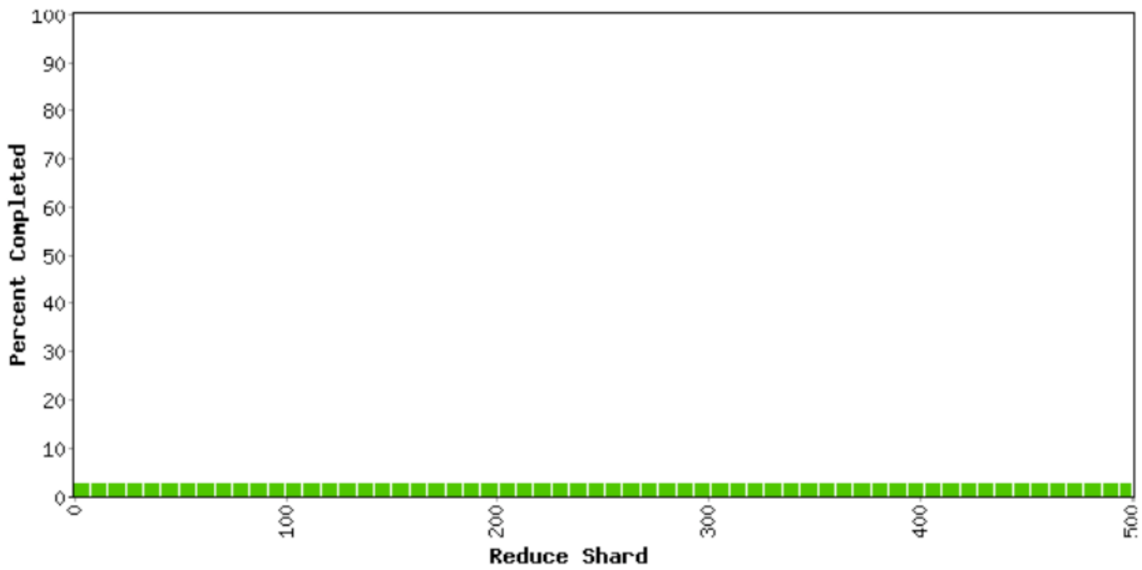


MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

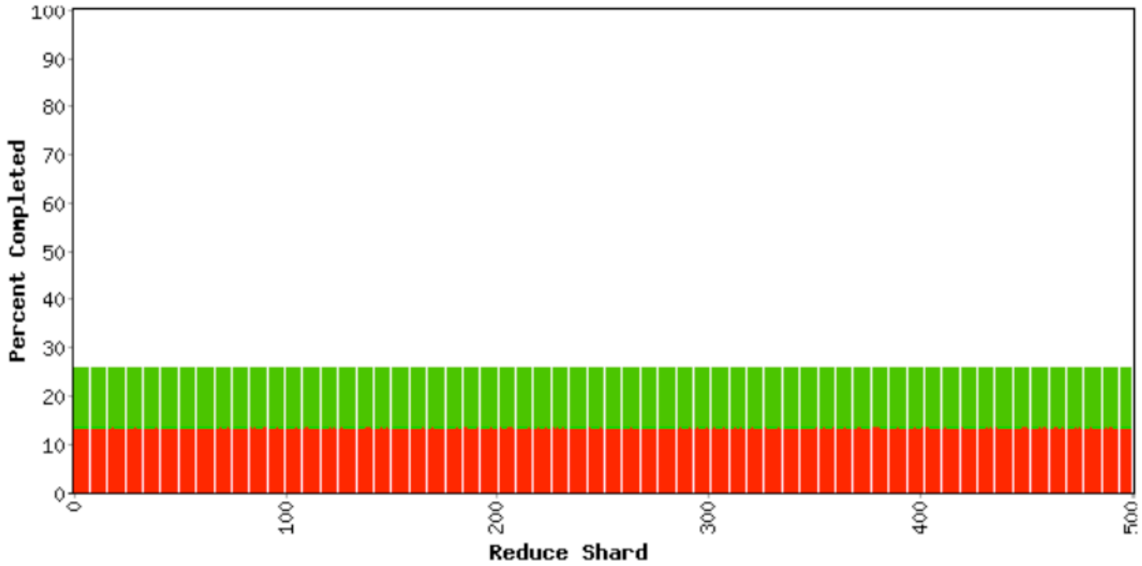
Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-outputs	506631

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

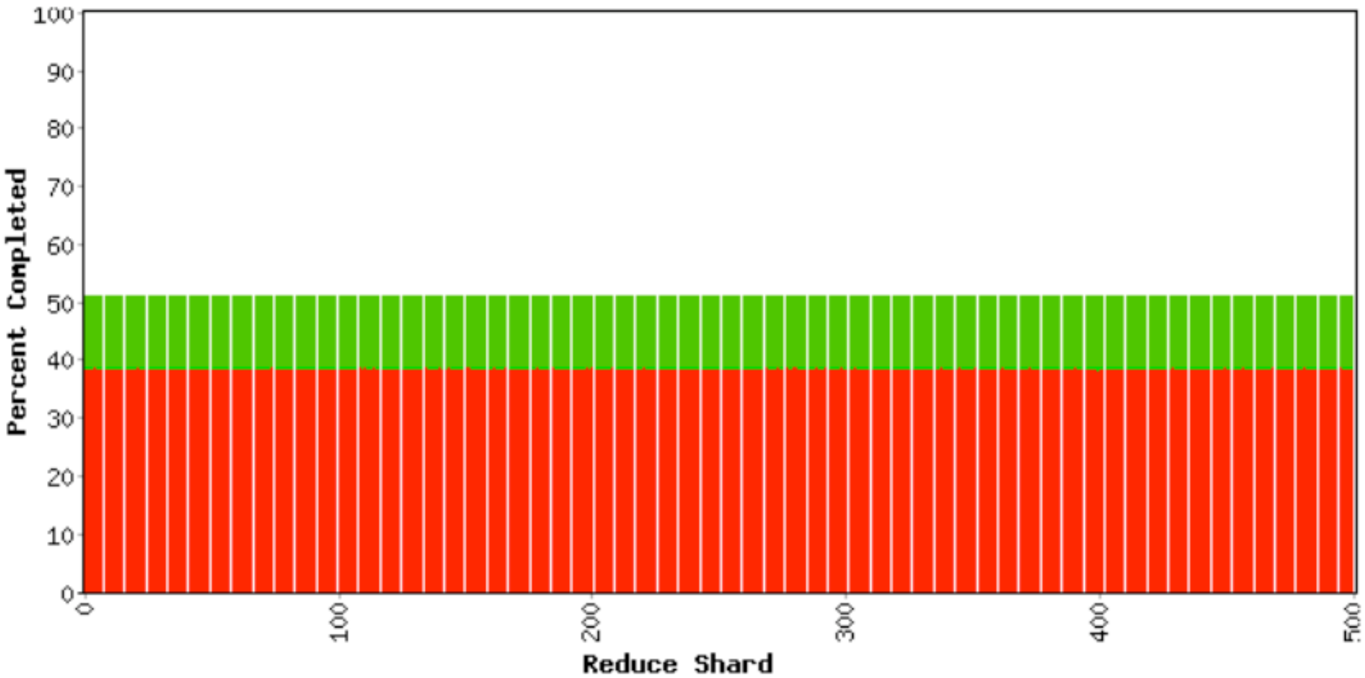
Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0

Counters

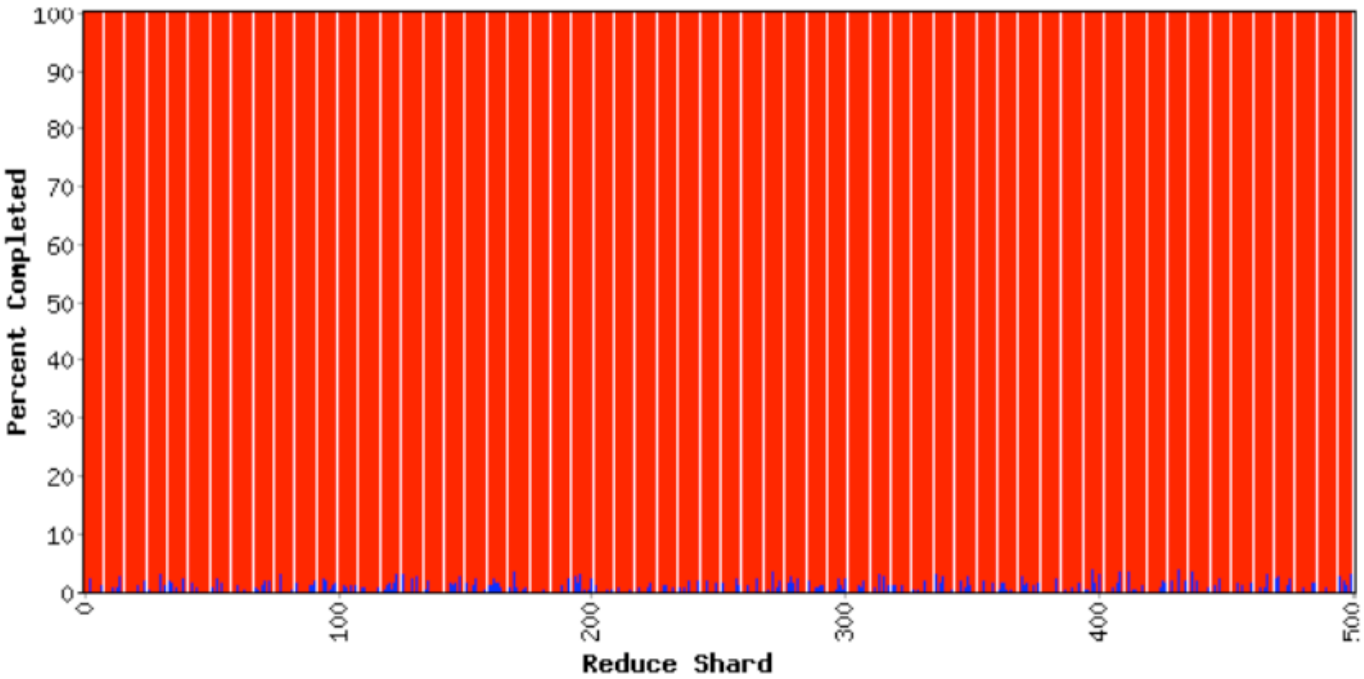
Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

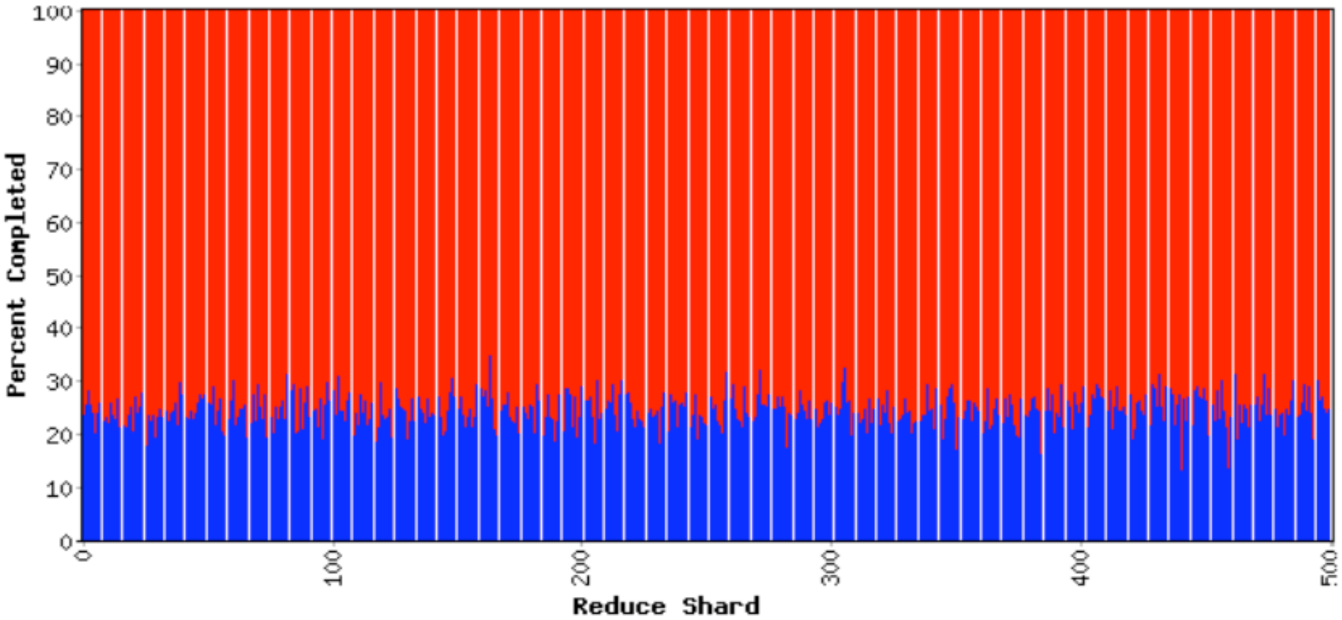
Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	105
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

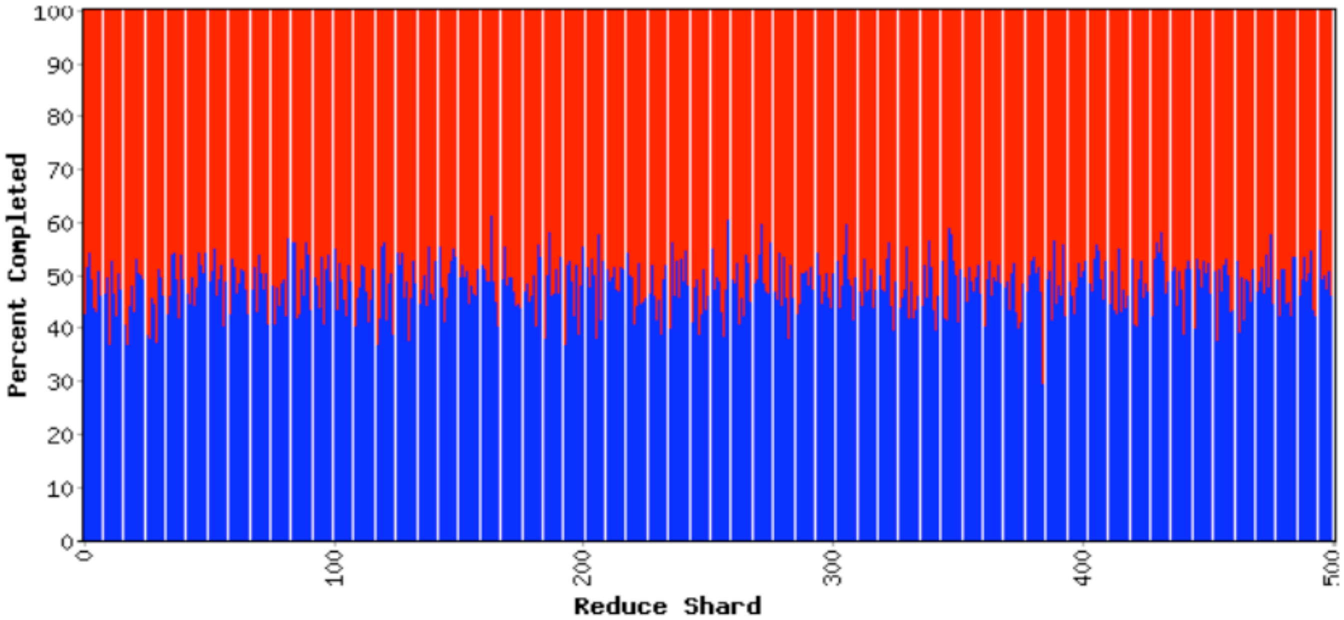
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.1	
Output (MB/s)	1238.8	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51738599	
mr-merge-outputs	51738599	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

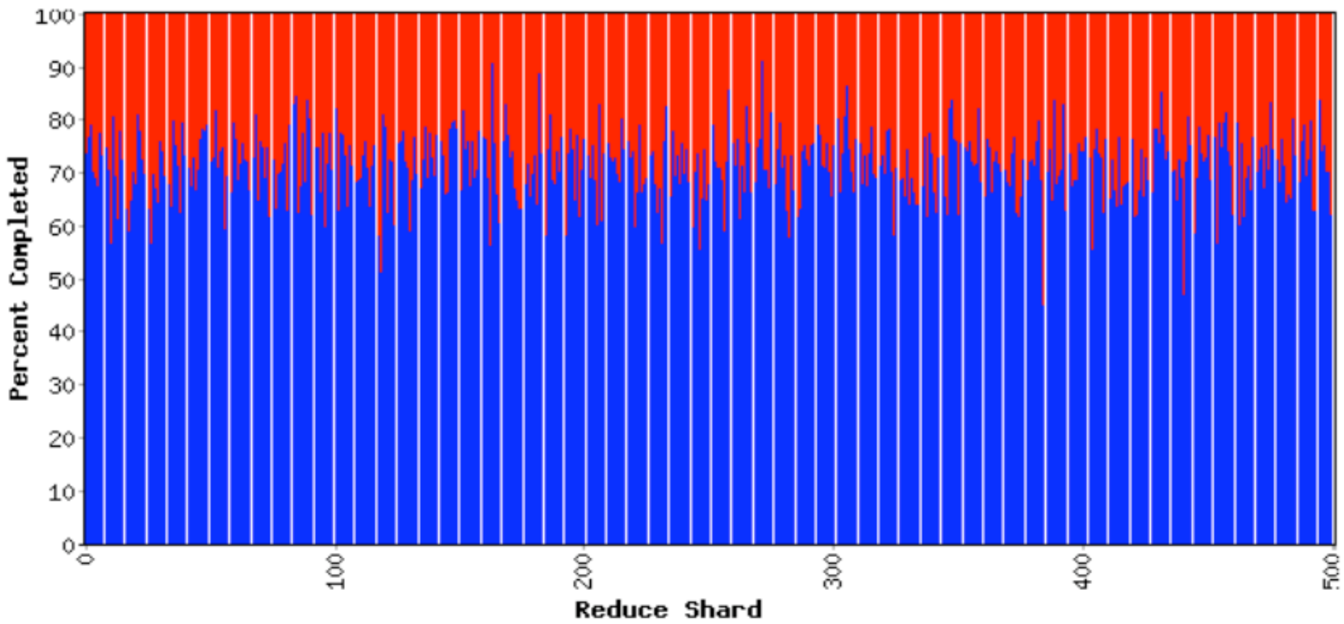
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1225.1	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51842100	
mr-merge-outputs	51842100	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

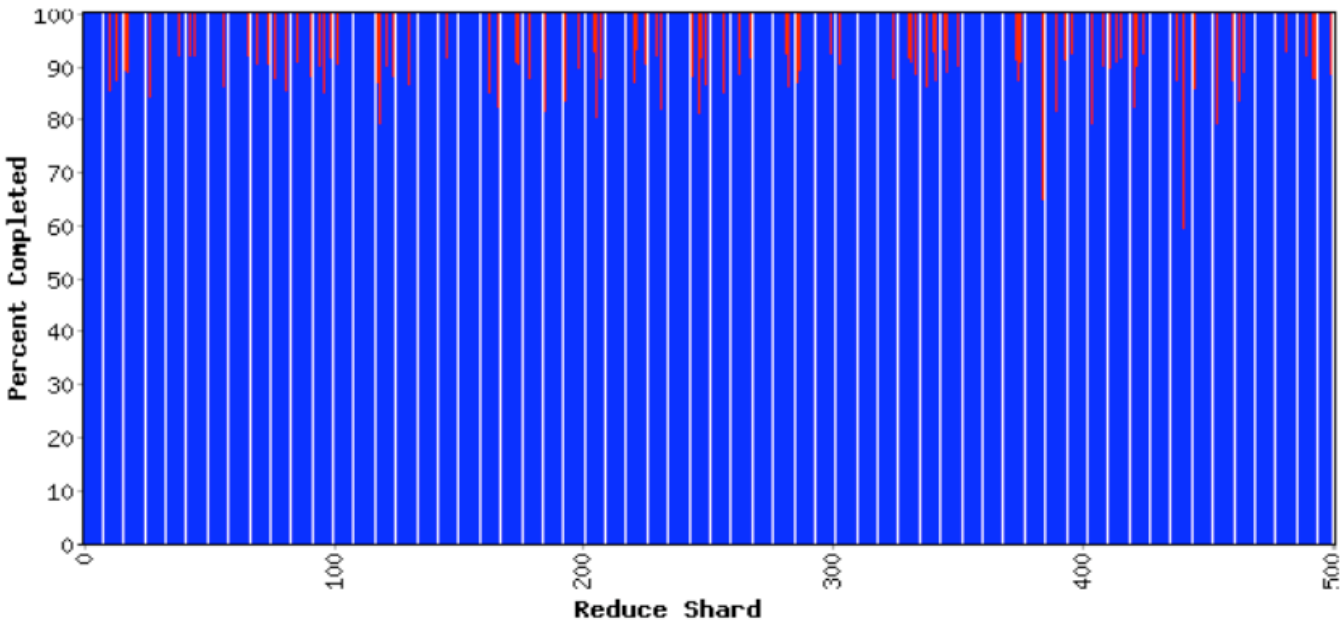
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1222.0	
doc-index-hits	0	1
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51640600	
mr-merge-outputs	51640600	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

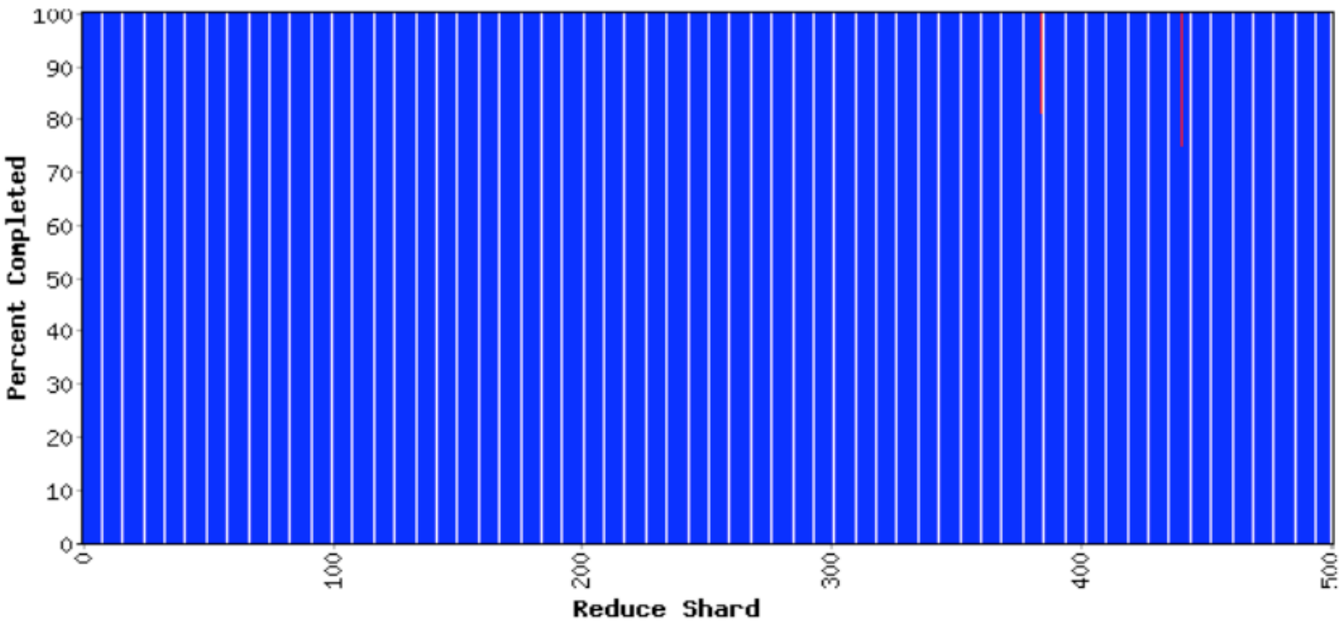
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

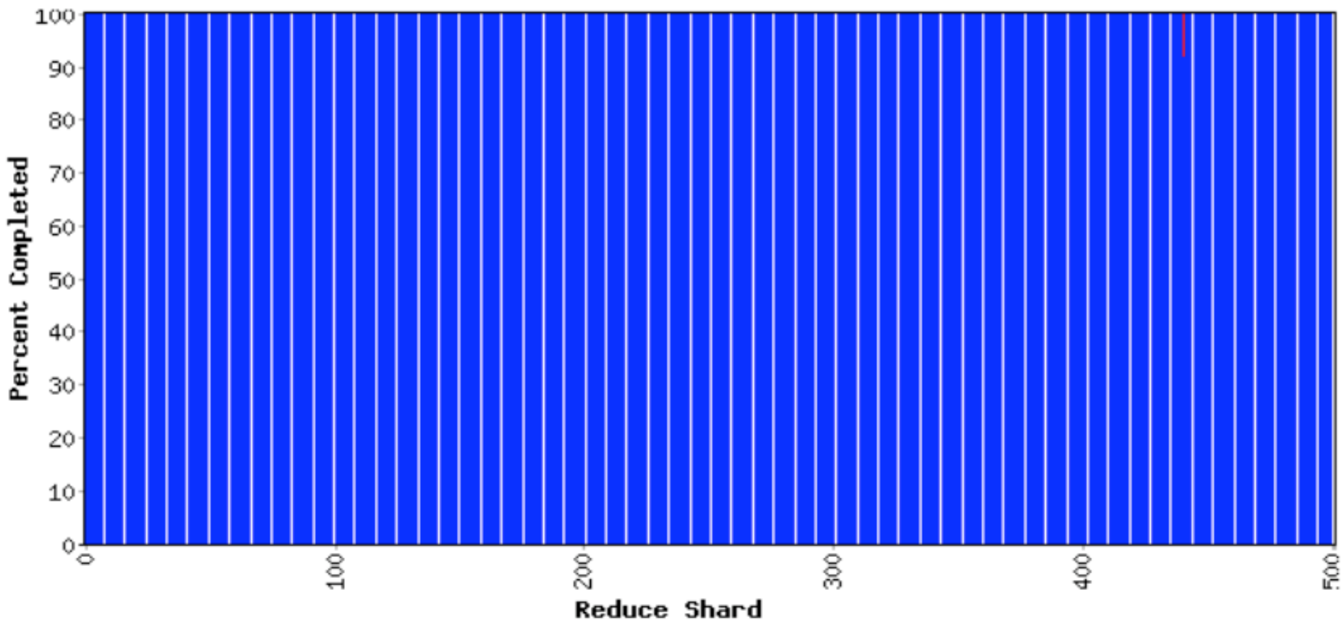
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	1056
docs-indexed	0	1
dups-in-index-merge	0	
mr-merge-calls	394792	1
mr-merge-outputs	394792	1

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

Refinement:

- Master scheduling policy:
  - Asks GFS for locations of replicas of input file blocks
  - Map tasks typically split into 64MB (GFS block size)
  - Map tasks scheduled so GFS input block replica are on same machine or same rack

- Effect
  - Thousands of machines read input at local disk speed
    - Without this, rack switches limit read rate

EC2 SOAP/Query API

- **Images:**
  - RegisterImage
  - DescribeImages
  - DeregisterImage
- **Instances:**
  - RunInstances
  - DescribeInstances
  - TerminateInstances
  - GetConsoleOutput
  - RebootInstances
- **Keypairs:**
  - CreateKeyPair
  - DescribeKeyPairs
  - DeleteKeyPair
- **Image Attributes:**
  - ModifyImageAttribute
  - DescribeImageAttribute
  - ResetImageAttribute
- **Security Groups:**
  - CreateSecurityGroup
  - DescribeSecurityGroups
  - DeleteSecurityGroup
  - AuthorizeSecurityGroupIngress
  - RevokeSecurityGroupIngress

CloudFront

Experience

Rewrote Google's production indexing System using MapReduce

- Set of 10, 14, 17, 21, 24 MapReduce operations
- New code is simpler, easier to understand
  - 3800 lines C++ → 700
- MapReduce handles failures, slow machines
- Easy to make indexing faster

Related Work

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as

Cloud versus the Grid

- Geographically distributed
- Across multiple administrative domains
- App's need high-level programming abstractions (e.g. workflow)

Steps

- Get Amazon account
  - http://www.amazonaws.com
- Boot instance of AMI image
- Log in with ssh
- Start Apache